NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ADVANCED QUALITY OF SERVICE MANAGEMENT FOR NEXT GENERATION INTERNET

by

Paulo R. Silva

September 2001

Thesis Advisor:	Geoffrey Xie
Second Reader:	Bert Lundy

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Advanced Quality of Service Management for Next Generation Internet	**5. FUNDING NUMBERS**
6. AUTHOR(S): Paulo R. Silva	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES): Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER:**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES): DARPA and NASA	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER:**

11. SUPPLEMENTARY NOTES: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution us unlimited.	**12b. DISTRIBUTION CODE** Statement A

13. ABSTRACT ***(maximum 200 words)***

Future computer networks, including the Next Generation Internet (NGI), will have to support applications with a wide range of service requirements, such as real-time communication services. These applications are particularly demanding since they require performance guarantees expressed in terms of delay, delay jitter, throughput and loss rate bounds. In order to provide such quality-of-service (QoS) guarantees, the network must implement a Resource Reservation mechanism for reserving resources such as bandwidth for individual connections. Additionally, the network must have an Admission Control mechanism, for selectively rejecting some QoS-sensitive flow requests based on resource availability or administrative policies.

The Server and Agent based Active network Management (SAAM) is a network management system designed to meet the requirements of NGI. In SAAM, emerging services models like Integrated Services (IntServ) and Differentiated Services (DiffServ), and the classical Best Effort service are concurrently sharing network resources. This thesis develops and demonstrates in SAAM a novel resource management concept that addresses the difficulties posed by QoS networks. With the new resource reservation and admission control approaches, the sharing mechanism is dynamic and adapts to network load. It ensures high resource utilization while meeting QoS requirements of network users.

14. SUBJECT TERMS: Next Generation Internet, Resource Management, Resource Allocation, Admission Control, Network Utilization, Quality of Service, Guaranteed Service, Integrated Service, Differentiated Service, Best Effort Service, Flows, Path Information Base.	**15. NUMBER OF PAGES:** 200
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

ADVANCED QUALITY OF SERVICE MANAGEMENT FOR NEXT GENERATION INTERNET

Paulo R. Silva
Lieutenant Commander, Portuguese Navy
B.S., Portuguese Naval Academy, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: [illegible]
Paulo R. Silva

Approved by: [illegible]
Geoffrey Xie, Thesis Advisor

[illegible]
Bert Lundy, Second Reader

[illegible]
Chris Eagle, Chairman
Department or Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Future computer networks, including the Next Generation Internet (NGI), will have to support applications with a wide range of service requirements, such as real-time communication services. These applications are particularly demanding since they require performance guarantees expressed in terms of delay, delay jitter, throughput and loss rate bounds. In order to provide such quality-of-service (QoS) guarantees, the network must implement a Resource Reservation mechanism for reserving resources such as bandwidth for individual connections. Additionally, the network must have an Admission Control mechanism, for selectively rejecting some QoS-sensitive flow requests based on resource availability or administrative policies.

The Server and Agent based Active network Management (SAAM) is a network management system designed to meet the requirements of NGI. In SAAM, emerging services models like Integrated Services (IntServ) and Differentiated Services (DiffServ), and the classical Best Effort service are concurrently sharing network resources. This thesis develops and demonstrates in SAAM a novel resource management concept that addresses the difficulties posed by QoS networks. With the new resource reservation and admission control approaches, the sharing mechanism is dynamic and adapts to network load. It ensures high resource utilization while meeting QoS requirements of network users.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Many people deserve a word of thanks for making this research possible. First, I appreciate the confidence Dr. Geoffrey Xie placed in me during the past year. His invaluable aid and mentoring fostered motivation and enthusiasm, greatly contributed to the successful completion of this research. It has been an absolute privilege to work with him. I also thank Mr. Cary Colwell for his availability and patient assistance rendered during this study. Thanks, also, to the entire SAAM team. The sense of camaraderie and teamwork gave purpose to this research. I am also very grateful to my colleagues at NPS, together with their families, helping to balance academic and personal needs. Most of all, I thank my wonderful wife Paula, and children Inês and Pedro, for they unconditional support, patience and love, especially during the last two years. They are my inspiration and the true focus of my work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Quality-of-Service (QoS) is definitely one of the most popular and challenging research topics in Internet computer networking nowadays. Today's Internet is an extremely versatile communication service for a wide range of applications. However, it is only aimed at providing best-effort (BE) service, where traffic is processed as quickly as possible, with no guarantees as to timeliness or actual delivery. Although this model perfectly fits many applications like e-mail or regular web browsing, it is commonly perceived that the best effort service cannot adequately support delay-sensitive and/or loss-sensitive applications, such as Internet telephony, multimedia conferencing, telemedicine and many others. These applications have in common the requirement for certain level of network QoS guarantees, measured by throughput, network delay and data loss rate.

In order to meet the QoS requirements of all potential traffic over the Internet, different approaches have been proposed. The Differentiated Services (DiffServ) approach is intended to provide a discrete number of service levels or classes, thus making it a scalable solution. Since DiffServ only prioritizes traffic among a limited number of classes, it does not truly provide for full QoS guarantees on a per user session basis. A different approach, the Integrated Service (IntServ) model is aimed to provide per-flow QoS guarantees, where a flow may represent the traffic generated by individual applications.

QoS guarantees can only be met if network resources such as link capacity and buffer space are previously allocated to requesting applications. Such a mechanism is called Resource Reservation. Networks must also have a way of selectively rejecting new flow requests based on resource availability or administrative policies. This mechanism is called Admission Control. Additionally, with the requirement to support multiple classes of service over the same infrastructure, networks have to provide a model for Link Sharing[1]. Collectively, resource reservation, admission control, and link sharing address

[1] Also termed Network Provisioning

the problem of Resource Management. Since applications with vastly different QoS demands will need to use the same infrastructure, resource management solutions must be flexible, adapting to different traffic mixes and load fluctuations. Moreover, a growing number of applications not only demand QoS guarantees but also generate an increasingly large volume of data, putting a huge load on current networks. Thus, another important objective of resource management is efficient use of resources.

Next Generation Internet (NGI) is one of several initiatives of the networking community to develop networks capable of both guaranteed and best effort services. The Server and Agent based Active network Management or SAAM, is an ongoing research project under the NGI initiative and it is sponsored by the Defense Advanced Research Projects Agency (DARPA) and National Aeronautics and Space Administration (NASA). SAAM implements and proves the feasibility of a server-based network management concept that addresses the resource management problem.

B. MILITARY RELEVANCE

The vision for future joint war fighting of US military is described in Joint Vision 2020 (JV2020) [1]. The concept of network-centric warfare (NCW), first conveyed in the JV2010 and carried forward in JV2020, represents a fundamental shift from the previous platform-centric warfare. Interoperability with external agencies and among forces of the allied nations is a growing necessity as recently proved with the combined NATO operations in the Balkans. Military operations in the current information age are organized around the NCW concept, through which information superiority translates into increased combat power. NCW is enabled by effectively networking sensors, decision makers and shooters to achieve shared awareness, increased speed of command and high levels of self-synchronization.

The NCW environment creates a wide range of network service requirements, only possible to meet through active and adaptive networks. SAAM is one of such networks being prototyped at the Naval Postgraduate School. Appendix A identifies some key enabler technologies of SAAM, which illustrate the importance of SAAM in the context of the NCW environment. The work of this thesis greatly contributes for the improvement of the SAAM concept and further extends the potential of SAAM to become a solution to the technical problems posed by NCW.

C. THESIS OBJECTIVES

The main objective of this research is to develop, implement and test a mechanism for managing the resources of quality of service capable networks, like SAAM. Special consideration is made with regard to multiple classes of QoS services sharing the same network infrastructure and the resource allocation mechanism for efficient utilization of network resources.

D. RESEARCH QUESTIONS

This thesis study provides answers for the following questions: (1) how can SAAM efficiently manage network resources while providing support for different classes of QoS traffic? (2) What is the impact of inter-service borrowing on the overall performance of QoS capable networks like SAAM?

E. SCOPE OF THE STUDY

Previous research efforts have already showed to be feasible the concept of a central network management authority for providing QoS guarantees to network users, as current SAAM prototype demonstrates. Cheng and Gibson [2], and Queck [3] created the foundations for a QoS management model. This thesis draws from their work, but is mainly focused on extending the potential of SAAM servers to efficiently and dynamically manage network resources while supporting different classes of guaranteed services. Specifically, the core data structure residing in the heart of SAAM, the Path Information Base (PIB), has been redesigned to provide support for a novel and more efficient resource management mechanism.

F. THESIS ORGANIZATION

The remainder of the thesis is organized into the following chapters:

- Chapter II – Background. This chapter introduces some of the underlying concepts related with quality-of-service and resource management in next generation networks. Additionally, a brief description of the SAAM concept is presented.

- Chapter III – Efficient Resource Management. This chapter details the development of the resource management mechanism required to manage resources of QoS capable networks like SAAM. In specific, a novel inter-service borrowing mechanism is presented.
- Chapter IV – Design. Details the design of the network management mechanism as applicable to SAAM.
- Chapter V – Implementation. Describes the implementation and integration details of the resource management mechanism developed in this thesis.
- Chapter VI – PIB Test and Performance Analysis. Describes the test and evaluation of PIB.
- Chapter VII – Conclusions and Recommendations. Concludes the thesis study with conclusions and the identification of areas of further study.
- Finally, several appendixes are included to support the thesis study.

II. BACKGROUND

Current trends in the development of real-time network applications indicate that the future Internet architecture will need to support a diversity of applications with different QoS requirements. Ongoing research on QoS has proven that enabling end-to-end QoS over the Internet introduces complexity in its overall functionality. Moreover, it affects areas like network management, business patterns of networking companies, and it also changes the way customer perceives the services offered by the network. Finding an efficient solution for end-to-end QoS over the Internet is not only one the most popular but also a very challenging research topic in computer networking today.

The current Internet architecture provides only simple services like IP addressing, routing, fragmentation and reassembling of IP datagrams. It relies on higher-level transport protocols for sequential and assured data delivery, and provides no guarantee as to timeliness and throughput of traffic. These services are widely known as best-effort services. Although these services are sufficient for traditional Internet applications like e-mail, web browsing or file-transfer, the same is not true for the emerging wave of applications like IP telephony, multimedia conferencing, or audio and video streaming. Consequently, the need to provide the current Internet with the mechanisms required to support QoS on the Internet is natural.

The SAAM project currently under development at the Naval Postgraduate School defines and implements a model of network management that provides a solution for the so-called Next Generation Internet (NGI). This chapter gives an overview of current Internet architectures related with providing QoS over the Internet, and how they fit the QoS requirement of next generation internet. Additionally, the SAAM model is briefly described.

A. QUALITY OF SERVICE ARCHITECTURES

The efforts to enable end-to-end QoS over the existing IP infrastructure, have led to the development of two different architectures: the Integrated Services (IntServ) architecture and the Differentiated Services (DiffServ) architecture. Although fundamentally different, both of these architectures are designed to support QoS over the Internet.

1. Integrated Service

The aim of the Integrated Service architecture is to provide customer on-demand QoS guarantees, e.g. bandwidth, delay and loss rate, and is ideally suited for real-time applications. The architectural design of IntServ is based on the notion that in order to fulfill the QoS requirement of the customers, network resources should be managed and controlled [4], thus implying that the admission control and resource reservation are the building blocks of this architecture. Event though IntServ provides the means for end-to-end QoS, it is still not widely implemented. Due to the maintenance of per-flow information, classification, reserving and management resources per-flow introduces complex scalability problems, especially at the core of high-speed networks where the number of flows to process is in the thousands to million ranges. Currently, IntServ has proven to be easily deployed only in small networks, where the number of IntServ flow is moderate and manageable.

2. Differentiated Services

The Differentiated Services architecture or DiffServ was developed to avoid the scalability problem and the complexity of IntServ. However, DiffServ only provides quality differentiation on traffic aggregates without strict guarantees on individual flows. This quality differentiation only offers to network users the guarantee that some traffic receives better service than others. The few predetermined levels of QoS are usually called traffic classes. As an example of a DiffServ model, classes may referred to as Gold, Silver and Bronze. Access control to service is regulated by pre-established service level agreements (SLA) between network access providers (ISPs) and their clients, which specify the service level agreed upon and the fees of the service.

B. RESOURCE MANAGEMENT

There are a number of emerging requirements for resource management in the Internet; these requirements include both link-sharing services and services for supporting QoS traffic. Link-sharing is required whenever network resources are to be shared among different agencies or traffic classes. Resource reservation and admission control are the base for providing QoS guarantees. All these three mechanisms play a vital role for the efficient management of resources in modern QoS networks.

1. Resource Reservation

As stated in [4], there is an inescapable requirement for networks to be able reserve resources, in order to provide QoS guarantees for specific user flows. The allocation of resources is typically done on a flow-by-flow basis as each new flow requests admission to the network. This in turn, requires flow-specific state information to be kept by routers along the path followed by the flow. In current Internet, based on the best-effort model, state information is only maintained by end applications. Such a stateless network is simple, robust and scales very well. A soft state approach for a QoS network would be desirable. Currently, the Resource Reservation Protocol (RSVP) is being used to support reservation of resources over an IP based network. It does so by simply providing a set of rules for the network and requesting applications to exchange information regarding the QoS requirement and admission and then to setup the required network resources.

2. Admission Control

Admission control is required to implement the decision logic that determines whether a new flow can be granted the requested QoS, without affecting guarantees already granted to previously admitted flows. In addition to ensuring that QoS guarantees are met, admission control may be used to enforce network administrative policies on resource reservations. Finally, admission control may also be an important tool on accounting and network administrative reporting. The simplest model of admission control would be the case in which the user asks for a specific QoS and the network either accepts or declines the request depending upon available resources. Since many applications can still be able to get acceptable service for different levels of QoS, the negotiation may often be more complex until a final flow spec is agreed upon.

As previously stated, the network is required to keep state information, i.e., remember the QoS parameters of past requests. One approach to admit a new flow would be to compute the worse case QoS bounds for each service based on such state information. A different approach, which is likely to provide better resource utilization, would rely on routers monitoring actual link usage by existing flows, and use this measured data as the basis for admission control. Although this approach as a higher risk of overload, it may yield more efficient link utilization. Furthermore, such a soft state approach may scale better for large-scale deployment.

3. Link Sharing

Link-sharing is a resource management mechanism created to manage network resources, such as bandwidth. Link-sharing relies on the basic assumption that bandwidth is a network resource that will always be limited. Some argue that when technologies like optic fiber be fully matured and widely adopted, bandwidth will no longer be of concern. However, the most commonly accepted counter-argument says that new applications will then exist to consume existing bandwidth. Link-sharing services are required whenever a network link is to be shared between agencies, protocol families, or service classes [5]. Multiple agencies may share the bandwidth of a link, where each one pays a fixed share of the costs, expecting to receive a guaranteed share of the link bandwidth. A second requirement is for link sharing of bandwidth on a link between protocol families. Controlled link sharing is desirable because protocols families have different responses to congestion. Another example for link-sharing is to share the bandwidth on a link between different classes of services, such as IntServ, DiffServ and Best Effort classes. All of the above three examples of link sharing explicitly deal with the aggregation of traffic on a link. While there are a number of different motivations for link-sharing in the network, the requirements for link-sharing are essentially the same, whether the link sharing is between organizations, service classes or families of protocols.

C. OVERVIEW OF SAAM

SAAM is an intelligent active network management system, developed to meet the requirements of next generation Internet. SAAM offers a solution to the problem of providing QoS guarantees while maintaining the simplicity, robustness and scalability of its underlying TCP/IP architecture. The SAAM model establishes a hierarchy of servers and routers that are grouped into hierarchically structured SAAM regions, as depicted in figure 2.1. SAAM servers assume the responsibility of all network management decisions within their own region, thus allowing for SAAM lightweight routers to focus only on traffic forwarding. This novel approach is the basis for implementing a complex service model that integrates multiple classes of QoS services. The service model for SAAM provides support for IntServ and DiffServ and with the current best-effort services.

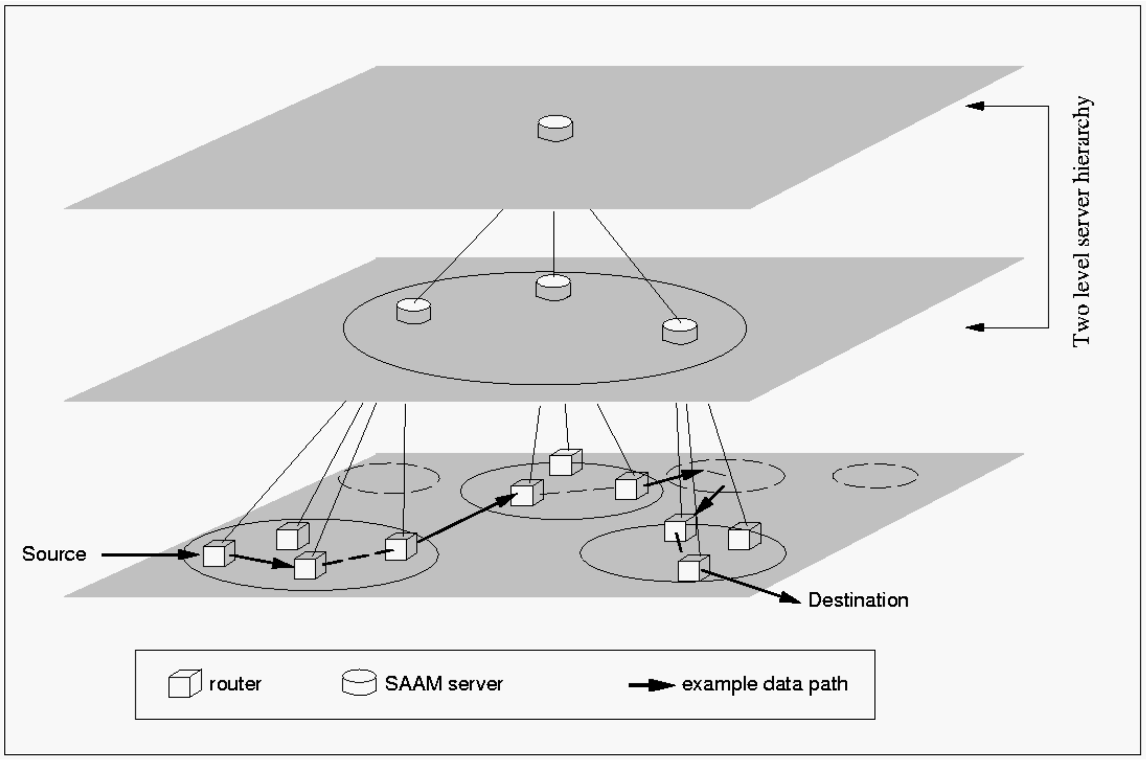


Figure 2.1 Hierarchical organization of SAAM.

1. SAAM Server

The SAAM server is a vital player within the SAAM architecture. The server dynamically builds and updates a knowledge base called Path Information Base (PIB) about its region upon receiving status information from its dependent routers. As the central repository of information for the SAAM region, the server is in a privileged position to make the best decisions in terms of granting network resources, optimizing network utilization, selection of the best routes, perform load balancing and any other network operational and administrative tasks, without having the burden of traffic processing only associated with the routers.

2. SAAM Router

Unlike standalone routers in existing IP networks, the SAAM router is conceptually simple and robust. Since all complex QoS routing decisions are taken by the server, the main task of the router is to process and forward traffic. Additionally, they monitor traffic and periodically report to the Server, the status of their links. The status information includes current link utilization, packet delay and data loss rate, on a per-service-class basis. Routing tables are updated by the server and contain information about the flow label and outbound interface to next router. SAAM routers are therefore required to maintain minimal state information, which provides for scalability.

3. SAAM Operation

The SAAM server initially discovers its region by flooding down the network with a special control message. Routers will in turn be aware of their server and report back their existence, which include all links between them and physical bandwidth available at their interfaces. The server uses this information to identify all paths within its region and to initially allocate bandwidth among all services classes supported, before starting accepting user traffic. Users request service at edge routers, which in turn will forward the request to the server. The admission control mechanism is then responsible for either accepting or rejecting the request based upon resource availability. Flow acceptance results in resources being allocated at each router along the selected path prior to send the new traffic. Edge routers maintain enough per-flow information, required for the purpose of traffic control and policing. At ingress or edge routers, packets are labeled to allow routers to forward their traffic. Additionally, IntServ packets will be inserted flow state information, which is then used and updated by the state-less high-speed core-routers when processing and forwarding traffic.

III. EFFICIENT RESOURCE MANAGEMENT

Chapter II introduced the concept of resource management and explained its key role for future networks. Admission control, resource allocation and link-sharing were identified as the building blocks for the activity of managing network resources in forthcoming QoS capable networks. This chapter builds on the previous chapter to describe an efficient and dynamic resource management model, which makes uses of those mechanisms to meet the requirements of the SAAM network. In providing support for multiple levels of QoS, e.g., those defined by the IntServ and DiffServ, a novel concept of inter-service borrowing of bandwidth will be presented as a solution to provide for a better resource management in such QoS networks.

A. RESOURCE MANAGEMENT APPROACHES

The current implementation of SAAM provides for multiple services classes, each offering one or more levels of services. These service classes, Integrated Service, Differentiated Service and Best Effort Service, will concurrently share network resources, e.g. link bandwidth. If the goal of efficient utilization of resources is to be met, then SAAM resource management must ensure that each service level takes an optimum share of network resources at any time, in order to maximize resource utilization. The sharing mechanism needs to be adaptive so that it adjusts to dynamic loads in each service class. It must also ensure that a minimum capacity exists at all times to fulfill the majority of future flow demands from higher priority service levels like IntServ while ensuring that lower service levels will not starve.

The complete link-share model for SAAM is depicted in Figure 3.1. In addition to the already mentioned IntServ, DiffServ and Best Effort service levels, the model also ensures that some bandwidth is reserved for signaling traffic, i.e. SAAM control traffic and finally, some other special traffic, named out-of profile (OP). OP traffic results from misbehaving user applications that generate traffic in excess of their previously negotiated QoS guarantees. In such cases, offending packets are marked as out-of-profile and pushed to the OP service level. Typically, OP packets receive the lowest priority. They are forwarded after all other traffic, i.e., when the link would be otherwise idle.

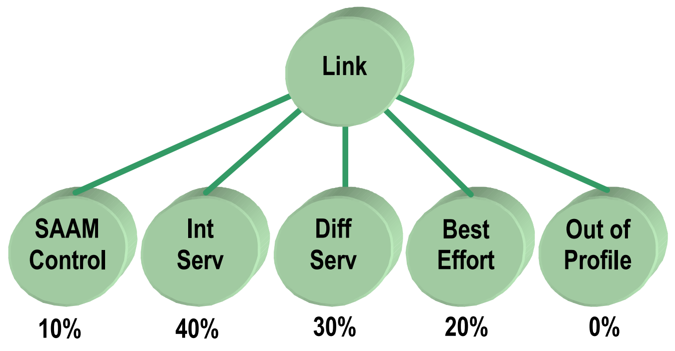


Figure 3.1 Link-share model in SAAM

1. Initial Resource Allocation

In the SAAM network, the SAAM server is responsible for the resource allocation. During initial startup and as part of the configuration cycle, participant routers advertise themselves and their interfaces to the SAAM server, which in turn develops the Path Information Base (PIB). The PIB is a data structure that will be used by the Server to determine all possible paths among all network nodes and to maintain link status information, as reported by routers. From the total bandwidth of each interface as reported by the hosting router, the Server then allocates a predetermined amount to each of the five service levels. This initial bandwidth is called base allocation per service level (BA) and is represented as a percentage of the total interface bandwidth. A typical link share per service level as previously used by SAAM is shown in Figure 3.2.

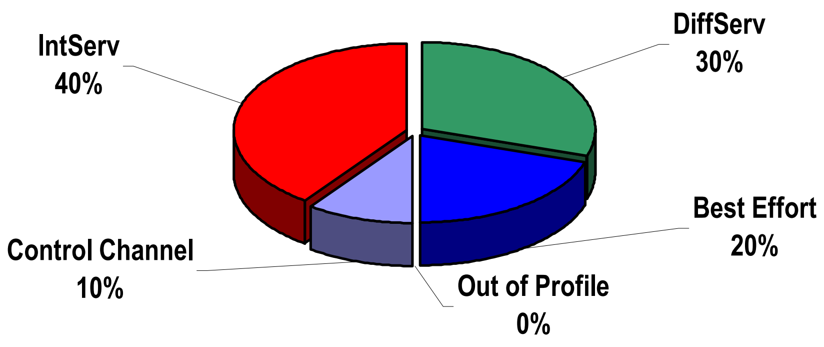


Figure 3.2 Typical link share percentages in SAAM

The share assigned for the SAAM control channels, is vital for the functioning of the entire SAAM region. As a goal in SAAM, the amount of control traffic should never be higher than the assigned 10% of the total bandwidth in each link. The non-conforming traffic of OP service pipe is typically given no bandwidth share. Consequently, OP packets are only served at each router when no other traffic is waiting for service. OP is the only service that may suffer from starvation when the network load is high. Finally, the share among the remaining three service levels shall then be assigned to fulfill the business and administrative goals of the networking service being provided, in an optimum and efficient way.

2. Developing an Optimum Share Model

Lets consider a single link with a maximum bandwidth capacity C_{max}, and with five service levels as mentioned before. Considering the individual capacities C_{CC}, C_I, C_D, C_B, C_{OP}, assigned respectively to SAAM control channels, IntServ, DiffServ, BE and OP service levels, the following expression holds:

$$C_{CC} + C_I + C_D + C_{BF} + C_{OP} \leq C_{\max} \tag{3.1}$$

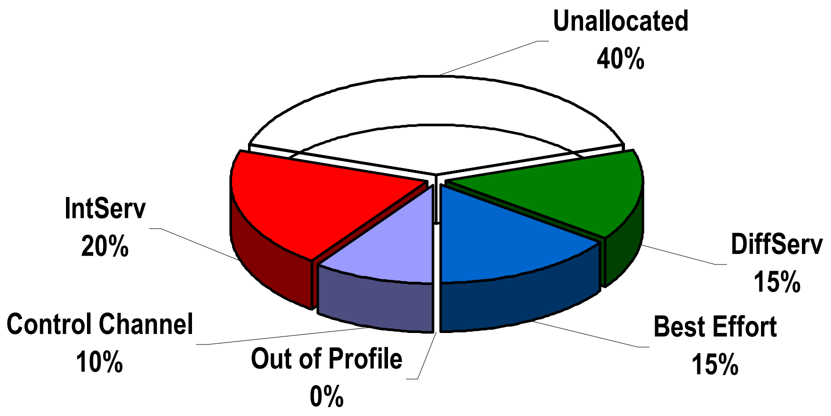


Figure 3.3 Link share model for SAAM with conservative base allocation

There are different alternatives for the distribution of the link share among the different classes of service. One of such alternatives could be performing static allocation, based on historical data and statistical models to forecast future load distribution for all service classes. While this solution requires the least effort after initial setup, it offers no flexibility and it is likely to result in poor network utilization. At some point, one service level may be using its entire allocated share and rejecting or buffering traffic while other services may be using a small fraction of their share, thus making an inefficient use of resources.

A different approach would be to make a conservative start by allocating little or minimum capacity for each of the service classes, which results in some of the link bandwidth not being assigned to any service – unallocated bandwidth (C_u). As new network users are admitted to the different classes of service, the utilization of each service level will then increase and eventually, reach the maximum initial capacity available for the class. From then on, the resource management mechanism ensures that new resource demands can be granted by claiming the initial unallocated capacity as long as there is still some available. The same way classes of service are allowed to claim unallocated bandwidth as needed, they should also release that bandwidth as soon as it is no longer required, thus making it available to other service classes. By dynamically resizing the capacity allocated for a service class, the network automatically adjusts to the profile of the traffic load, which increases the overall network utilization.

The previous approach may be further extended to achieve event better network utilization. In some cases, a heavily loaded service level (say IntServ) may be using all of its base allocation and the entire link unallocated capacity while another service class (say DiffServ) is using only a small fraction of its base allocation. Intuitively, if the possibility for a dramatic increase of DiffServ traffic in the near future is remote, then some fraction of DiffServ's unused base allocation bandwidth could be made available for borrowing by IntServ. This solution would yield even better link bandwidth utilization and decrease the total number of rejections of user flow requests, especially under high network loads. This novel approach is named Inter-service Borrowing and will be explained in detail in the following sections.

B. CONSERVATIVE BASE ALLOCATION

The previous section introduced the concept of a conservative start for allocating bandwidth, which in turn leads to some fraction of the link capacity not being allocated.

The initial inequality as expressed in equation 3.1 should be modified to include the unallocated capacity, thus turning the initial equation into

$$C_I + C_D + C_{CC} + C_{BF} + C_{OP} + C_U = C_{\max},$$

where C_u refers to the unallocated capacity.

At this point, it is of interest to differentiate the concept of initial allocation (base allocation) and current allocation. Base allocation will be denoted by C_{Xo} (for service class *X)* and is a constant value that is assigned on network startup as opposed to C_X which is the current allocated value for service *X*. The current allocation value varies over time, depending on the load of the respective service, and is initially set equal to the base allocation C_{Xo}, i.e., at the startup the following two equalities exist:

$C_{I_o} = C_I$ for IntServ

$C_{D_o} = C_D$ for DiffServ

According to the SAAM model, some of the services will have a constant share of the link bandwidth. We will therefore assume that SAAM control channel and Best Effort services will have a fixed share allocation over time[2]. Additionally, Out of Profile traffic will have no capacity specifically allocated and therefore $C_{OP} = 0$.

Based on the above assumptions, the initial setup of the network is therefore performed in accordance with the following equation:

$$C_{Io} + C_{Do} + C_{CC} + C_{BF} + C_U = C_{\max}. \tag{3.2}$$

With the assumption that C_{CC} and C_{BF} remain constant over time, let us isolate three dynamic terms of interest of this discussion. It will be,

$$\begin{aligned} C_{Io} + C_{Do} + C_U &= C_{\max} - (C_{CC} + C_{BF}) \\ C_{Io} + C_{Do} + C_U &= kC_{\max} \\ C_{Io} + C_{Do} + C_U &= C'_{\max} \end{aligned}$$

IntServ and DiffServ classes of service will use network resources on demand. The initial capacity allocated for these two classes (C_{Io} and C_{Do}) may be used either

[2] An alternative for BE in SAAM, may be the utilization of unused capacity allocated for Control Channels, IntServ and DiffServ. This could be achieved with a work-conserving packet schedule algorithm.

partially or totally. Whenever any of these classes use their base allocation share, they become eligible for taking some of the unallocated capacity as required. Eventually, no more unallocated space is available, which is then the case to try inter-service borrowing, between these two classes. The next section will address this sequence in detail.

C. ADMISSION CONTROL SEQUENCE

As suggested in the previous section, the process of dynamic resizing the bandwidth assigned to a service level goes through a sequence of different phases. This readjusting process is triggered by the arrival of a new IntServ or DiffServ flow request and is part of the admission control mechanism. The admission control procedure can be divided in three distinct steps, each corresponding to a distinct admission phase, as illustrated in figure 3.4, and they are:

- Step 1 - Direct admission
- Step 2 – Dynamic growing
- Step 3 – Inter-service borrowing

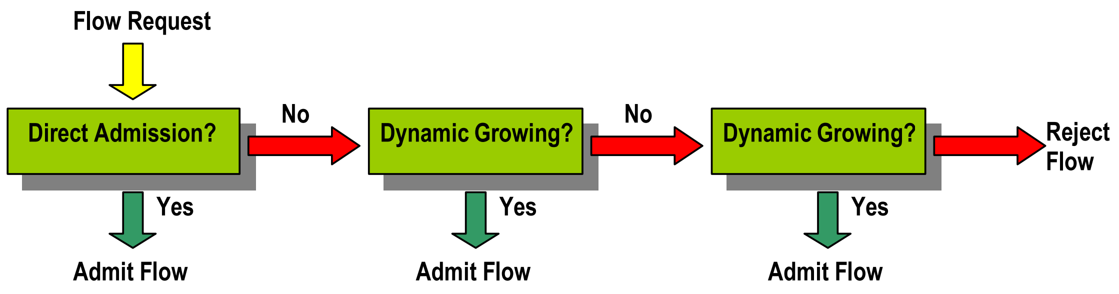


Figure 3.4 Admission control sequence.

1. Direct Admission – Admission Step 1

Direct admission is the first phase of the admission. It is the most basic admission control step, because a single service class is considered. A new flow is admitted at this point if there is enough available bandwidth from the initial base allocation for that service. All new flow requests go through this phase. If admission fails at this stage, it means the service load has increased over the initial base allocation for the service and the flow request must therefore proceed to phase 2. Recall that variables C_I and C_D represent the current allocated capacity for IntServ and DiffServ, respectively. As already stated, initially we have:

$$C_I = C_{Io} \text{ and } C_D = C_{Do}$$

Let's now consider the sequence of steps required to admit a new flow request. For simplicity and without loss of generality, it is assumed that new flow request belong to IntServ. Only two classes of services will be considered: IntServ and DiffServ. However, the following deduction could easily be extended to *n* different classes of services. Considering that a new flow request f^* arrives and has a bandwidth requirement of R_{f^*}, the admission criteria for this flow is based on the equation

$$\sum_{f \in B_I} R_f + R_{f^*} \leq \alpha_I C_I \tag{3.3}$$

where B_I is the set of currently active flows of service I (IntServ).

Equation 3.3 introduces also the factor α_I, which is referred to as the load factor for IntServ. Typical network load management uses this load factor to prevent maximum load to approach the full physical link capacity. Although the actual value for this factor is not relevant for the present discussion, it is included in the expressions herein presented for completeness. Since $0 < \alpha_I \leq 1$, we can at most use $\alpha_I C_I$ capacity which is less than or equal to C_I. Whenever inequality 3.3 holds then the new flow f^* may be admitted and the admission procedure does not need to go into next step.

2. Dynamic Growing – Admission Step 2

Dynamic growing refers to the ability for the link share mechanism to dynamically adapt to changing service level load. By starting with a conservative small base allocation of bandwidth for each of the service levels, this step ensures that each service is allowed to received more resources from the unallocated capacity, only when they need them. Inversely, when those resources are no longer required for that service, they are released, thus made available for other services. Suppose inequality 3.3 is not satisfied during the previous step. It means that there is not enough capacity C_I to satisfy the new request. This next step now will attempt to use some of the possible unallocated capacity. The admission condition then becomes

$$\sum_{f \in B_I} R_f + R_{f*} \leq \alpha_I \left(C_I + \beta C_U\right) \tag{3.4}$$

The difference from equation 3.3 is that we now try to increase C_I by a small amount, enough to accept the new flow request. Intuitively, we need both

$C_U > 0$ and $\beta > 0$

The amount of unallocated bandwidth claimed to satisfy the new request should be minimum. Therefore, from inequality 3.4 we can derive the optimal value for β as follows:

$$\sum_{f \in B_I} R_f + R_{f*} = \alpha_I C_I + \alpha_I \beta C_U \tag{3.5}$$

$$\beta = \frac{\sum_{f \in B_I} R_f + R_{f*} - \alpha_I C_I}{\alpha_1 C_U}$$

The new flow may then admitted if:

$C_U > 0$

and

$0 < \beta \leq 1$

In the case of $\beta = 1$, the new flow is admitted taking all of the unallocated capacity. After admission, the bandwidth allocation for IntServ is increased by a given amount and the unallocated capacity is decreased by the same amount. Respective variables must be updated in the following order:

$C_I = C_I + \beta C_U$

$C_U = C_U - \beta C_U$

By doing this, we are enabling dynamic growing of the total link share for a given class of service, which may happen as long as there is enough unallocated space.

Whenever it is no longer possible to dynamically grow this way, the admission procedure proceeds to step 3.

3. Inter-service Borrowing – Admission Step 3

Inter-service borrowing is to be considered only when the load of one service level peaks and other services are not using their entire base allocated bandwidth. When the load of a given service level is very high, it is possible that the sum of the base allocation with the unallocated space is not enough to fulfill the resource demand for that service. In such case, the admission process fails both phase 1 and phase 2 steps. However, under some circumstances it might be possible to borrow some capacity from other services. This inter-service borrowing mechanism is based on a statistic model for the network traffic and will be explained in the next section. For now, it matters only to focus on the new admission condition applicable to this last case.

The admission condition for phase 2 - equation 3.4 - now no longer holds. Providing that some capacity can be borrowed from some other service, that borrowed bandwidth should be added to the right side of equation 3.4, which then becomes:

$$\sum_{f \in B_I} R_f + R_{f*} \leq \alpha_I \left(C_I + \beta C_U + \rho_D C_D \right) \tag{3.6}$$

In equation 3.6, C_I refers to the current total allocation for the service I (IntServ), βC_U is the available unallocated capacity, and finally, $\rho_D C_D$ represents the maximum bandwidth that service *D* (DiffServ in this case) makes available for inter-service borrowing. It is important to note that at this stage in the admission process, the unallocated capacity C_U is either zero or very small, and is insufficient to meet the new flow requirement. That remaining capacity should therefore be completely used before going into inter-service borrowing, which explains the βC_U in equation 3.6. For this reason, $\beta=1$ and the admission condition can me modified to become:

$$\sum_{f \in B_I} R_f + R_{f*} \leq \alpha_I \left(C_I + C_U + \rho_D C_{D_o} \right) \tag{3.7}$$

This new equation differs from equation 3.5 by the new term $\rho_D C_{Do}$, where C_{Do} is the base allocation for service D (DiffServ) and ρ_D the fraction of that capacity that might be borrowed. Since the base allocation for DiffServ is constant, the solution for the inequality depends upon the value of ρ_D. As already stated, next section will detail a way of calculating ρ_D. For now, assume that we have obtained a value for ρ_D. With this value, either the equation does not hold and the new flow request is rejected, or instead, the new flow can be admitted.

If the new flow is accepted, inter-class borrowing takes place. For instance, using the previous example, it would mean that DiffServ has its available bandwidth partially reduced until resources are released. In order to keep track of this capacity transfer, variables C_I and C_D must be updated to reflect the new allocation transfer. The bandwidth transfer from DiffServ to IntServ in this example should be no more than the necessary to admit the new flow. This means that we are interested in the minimum value for ρ_D that satisfies the inequality 3.7, i.e.

$$\rho_D' = \min(\rho_D)$$

such that,

$$\sum_{f \in B_I} R_f + R_{f*} = \alpha_I (C_I + C_U + \rho'_D C_D) \tag{3.8}$$

Expressing ρ' as the dependent variable, equation 3.8 becomes:

$$\rho_D' = \frac{\sum_{f \in B_I} R_f + R_{f*} - \alpha_I C_I}{\alpha_1 C_D} - \frac{C_I + C_U}{C_D} \tag{3.9}$$

After the admission of the new flow, the allocation variables should be updated as follows:

$$C_I = C_I + C_U + \rho_D' C_D$$

$$C_U = 0$$

$$C_D = C_D - \rho_D' \mathrm{C_D}$$

D. INTER-SERVICE BORROWING EXPLAINED

If the network load were easily predictable and constant, it would be possible to adjust the resource shares of different service levels, in an optimum way, such that the resource utilization would be maximum and the flow rejection rate minimum. However, in the real world, this is never the case and there is always some uncertainty on traffic prediction. Traffic load fluctuations are more than likely and therefore the resource management algorithm should be dynamic and adaptive, in order to maximize resource utilization and ensure availability, even when network load is unbalanced among service levels. Inter-service borrowing seems a logic step in this regard.

1. Aggregated Flow Distribution

As discussed in the previous section, the three-step admission procedure relies on inter-service borrowing at last, to admit a new flow at extreme load levels. The admission condition for inter-service borrowing (equation 3.7) shows that the admission decision is a function of the coefficient ρ, defined as the required fraction of the base bandwidth allocation of the service level to borrow from. A service should make available some of its base allocation capacity only when there is a minimal probability for that service to use that capacity in the near future. The basic idea is to obtain some quantitative measure of the bandwidth that a service can make available for borrowing, without reducing much of its own ability to accept new flows.

Consider that network clients request flows with different throughput requirements. We may assume that the throughput requirements of individual flows follow some type of unknown distribution with the following characteristics:

μ_O - mean value of an individual flow request

σ_O - standard deviation of the flow request distribution

If we now consider that ($R_1, R_2, R_3, \ldots, R_n$) is a random sample from the above distribution and that they represent at any time, the individual throughputs of n active flows, then:

$\overline{R}$ is the mean value of the sample.

According to the Central Limit Theorem [6], $\overline{R}$ is approximately a normal distribution with:

$E(\overline{R}) = \mu_{\overline{R}} = \mu_o$ (mean)

$V(\overline{R}) = \sigma_{\overline{R}}^2 = \frac{\sigma_o^2}{n}$ (variance)

$\sigma_{\overline{R}} = \frac{\sigma_o}{\sqrt{n}}$ (standard deviation)

The total throughput T of the set of n active flows is the sample total and can easily be obtained by the equation bellow:

$$T = \sum_{i=1}^{n} R_i$$

and

$E(T) = nE(\overline{R}) = n\mu_o$ (expected mean value for total throughput)

$V(T) = nV(\overline{R}) = n\sigma_o^2$ (expected variance)

$\sigma_T = \sqrt{n}\sigma_o$ (standard deviation)

2. Probabilistic Bandwidth Utilization

Now consider that the number of flows n is constant and that the throughput requests of individual flows follow the same distribution with mean μ_T and standard deviation σ_T. For any given probability p, we are now interested to obtain a value x such that

$$P(T \leq x) = p\,.$$

So, it will be

$$P\left(Z \leq \frac{x - \mu_T}{\sigma_T}\right) = p$$

$$\Phi\left(\frac{x - \mu_T}{\sigma_T}\right) = p$$

$$\Phi\left(\frac{x - n\mu_o}{\sqrt{n}\sigma_o}\right) = p$$

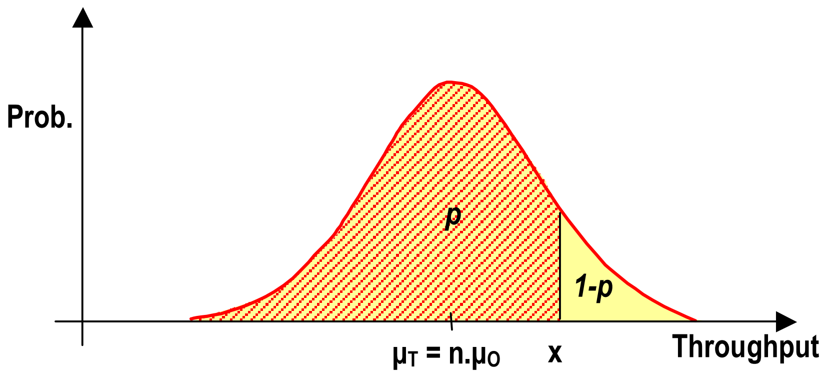


Figure 3.5 Aggregated throughput distribution as a normal distribution. Total throughput T of all n active flows is less than x with a probability p.

As an example, lets consider a probability *p = 0.9495 ≈ 95%.*

We then have

$$\Phi\left(\frac{x - n\mu_o}{\sqrt{n}\sigma_o}\right) = 0.9495$$

$$\frac{x - n\mu_o}{\sqrt{n}\sigma_o} = 1.64$$

$$x = n\mu_o + 1.64\sqrt{n}\sigma_o \tag{3.10}$$

or

$$x = \mu_T + 1.64\sigma_T \tag{3.11}$$

Equations 3.10 and 3.11 allow us to obtain a value of *x* based on the probability of 95%. Different values could be selected. Depending on what data is available, either equation might be used. For instance, if we precisely know the mean flow request, the standard deviation of the distribution of those flow requests and the number of active flows, equation 3.10 is to be used. If otherwise we know nothing about the original flow distribution or if the number of flows is not exactly known, we may otherwise have to use equation 3.11.

3. A Scalable Soft-State Solution for SAAM

One of the underlying goals of SAAM is to keep the minimal network status information, i.e. maintain a soft state, thus providing for robustness and scalability, without too much increase of complexity. For instance, once new flows are admitted, no information is kept with regard to the actual number of active flows. In equation 3.10, x is expressed as a function of the number of active flows and therefore this equation cannot be used in SAAM. The only alternative is equation 3.11, which expresses x as a function of the two parameters μ_T and σ_T, that characterize the distribution of the total throughput T.

We can estimate μ_T with

$$\mu_T = \sum_{f \in B} R_f \ ,$$

where B is the set of currently active flows of this service class. Equation 3.11 now becomes

$$x = \sum_{f \in B} R_f + 1.64\sigma_T \tag{3.12}$$

The SAAM server continuously receives link state information from routers. This information contains not only bandwidth utilization per class of service but also other QoS metrics like data delay and data loss rate. This soft state approach decreases the complexity at the core routers because no state information is maintained at flow level. No mechanism to explicitly notify network about termination of a flow exists. In fact, once a new flow is admitted and resources are reserved accordingly, the process of releasing those resources is implicitly part of the periodic link state advertising. The central SAAM server only maintains updated information about the throughput of the set of active flows for each router interface. Consequently, the term $\sum_{f \in B} R_f$ is easily obtained for every class of services. However, in 3.11 we still have the unknown standard deviation parameter of the flow aggregate distribution. With the knowledge of the number of active flows and the characteristics of the individual flow request distribution, it would be straightforward to compute the standard deviation. However, since we assumed to know nothing about the number of flows nor about the original distribution, a different approach must be followed.

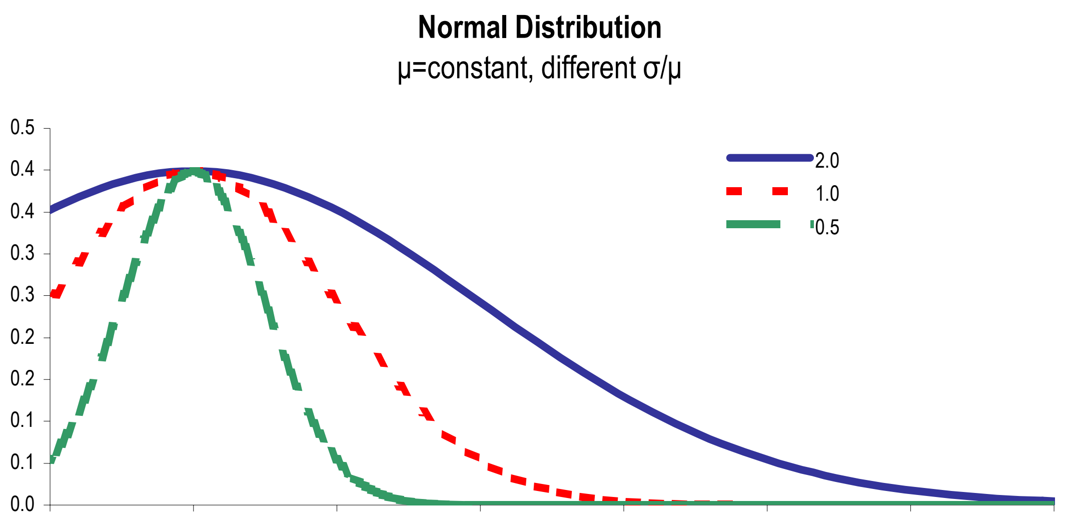


Figure 3.6 Normal distribution for different three ratios σ/μ.

It was already shown that the total throughput distribution and the individual flow request distribution are related with each other and that the following equations apply:

$$\sigma_T = \sqrt{n}\sigma_0 \tag{3.13}$$

and

$$\mu_T = n\mu_0 \tag{3.14}$$

Dividing equation 3.13 by equation 3.14 we obtain

$$\frac{\sigma_T}{\mu_T} = \frac{\sqrt{n}\sigma_0}{n\mu_o}$$

$$\frac{\sigma_T}{\mu_T} = \frac{1}{\sqrt{n}}\frac{\sigma_0}{\mu_o} \tag{3.15}$$

It is reasonable to assume that some information is known about the profile of the individual flow requests. From past data, there should be at least knowledge about the mean value of the bandwidth request per flow. However, we will take a conservative approach by assuming that we only know something about the shape of the flow request distribution, i.e. the ratio σ_o/μ_o.

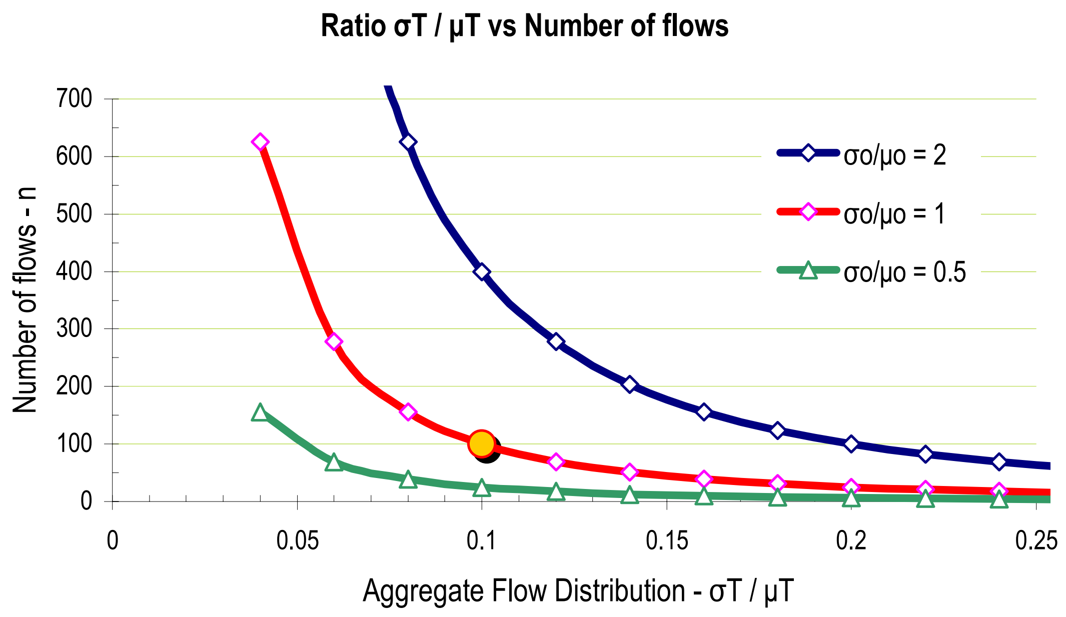


Figure 3.7 Ratio σ_T/μ_T versus the number of participant flows for three different shapes of flow request distributions.

Figure 3.7 shows a graph plotted with equation 3.15. For the aggregated throughput distribution, the number of flows as a function of the ratio σ_T/μ_T, for three different shapes of the original flows request distribution, i.e. for the ratios $\sigma_o/\mu_o = 2$, 1 and 0.5.

Lets now assume that the standard deviation of the original flow distribution is equal to its mean, i.e. $\sigma_o/\mu_o = 1$. This is considered a conservative approach since the real distribution is likely to be much more concentrated about the mean, i.e. with a ratio $\sigma_o/\mu_o < 1$. Additionally, it is assumed a substantial number of active flows, for example n = 100. Referring to the graph in figure 3.7, these assumptions apply to the (0.1,100) coordinate point, marked with a big circle. By inspecting the graph, it can be observed that as the number of flows increases or the σ_o/μ_o ratio decreases (narrow original flow distributions) the ratio σ_T/μ_T, also decreased. With the given assumption, equation 3.14 becomes

$$\frac{\sigma_T}{\mu_T} = \frac{1}{\sqrt{100}} \times 1 = 0.1$$

$$\sigma_T = 0.1\mu_T$$

We can now replace σ_T in equation 3.12, which becomes

$$x = \sum_{f \in B} R_f + 1.64 \times 0.1\mu_T$$

and finally conclude with:

$$x = \sum_{f \in B} R_f + 0.164 \sum_{f \in B} R_f$$

or

$$x = 1.164 \sum_{f \in B} R_f \qquad (3.16)$$

In summary, when the number of active flows equal or exceeds 100, and the original flow request distribution satisfies $\sigma_o \leq \mu_o$, we can ensure with a certainty of 95% that the throughput requirement of this service class will not exceed the value x as given by equation 3.16. Figure 3.8 is a pictorial representation of this concept applied to the DiffServ class.

As discussed in the previous section, the admission condition for inter-service borrowing was given by

$$\sum_{f \in B_I} R_f + R_{f*} \leq \alpha_I \left(C_I + C_U + \rho_D C_{D_o} \right)$$

The term $\rho_D C_D$ represents the maximum capacity that can be borrowed, can now be expressed as follows:

$$\rho_D C_D = \alpha_D C_D - x$$

$$\rho_D C_D = \alpha_D C_D - \left(\sum_{f \in B_D} R_f + 0.164 \sum_{f \in B_D} R_f \right)$$

$$\rho_D C_D = \alpha_D C_D - 1.164 \sum_{f \in B_D} R_f$$

and the complete admission equation becomes

$$\sum_{f \in B_I} R_f + R_{f*} \leq \alpha_I \left(C_I + C_U + \alpha_D C_D - 1.164 \sum_{f \in B_D} R_f \right) \tag{3.17}$$

Equation 3.16 is therefore applicable for a probability of 95% and a number of active flows $n \geq 100$.

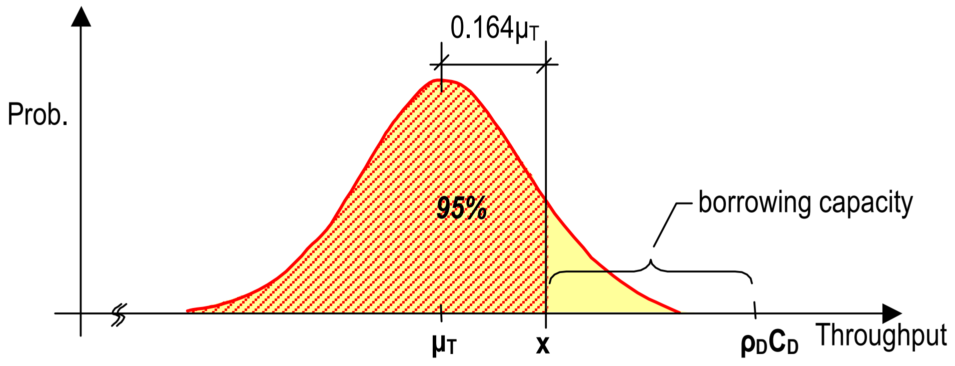


Figure 3.8 Pictorial representation of the borrowing capacity of DiffServ, based on the probability of 95%.

E. CHAPTER SUMMARY

QoS capable networks and the consequent need for resource allocation present new challenges for the efficient use of network resources. The link-sharing mechanism allows multiple service levels to share the same link, however in providing such versatility, dynamic and adaptive share mechanisms must be used in order to preserve high levels of resource utilization. The concept of resource management described in this chapter, allows for multiple services levels to adjust their link share by dynamically resizing their allocated capacity and using the unallocated capacity until they reach an point of equilibrium, which self adjusts to changing network load conditions. Moreover, by means of inter-service capacity borrowing, it is possible to further enhance resource utilization. It should be understood, however, that the complexity associated with this resource management approach only makes sense because it is assumed that network resources are limited and we want to maximize resource utilization levels. Next chapter will detail the implementation of all these concepts in the current SAAM prototype.

IV. SAAM QOS MANAGEMENT DESIGN

In the previous chapter, the theory behind the new advanced resource management concept was explained in detail. As the main component of the SAAM server, the Path Information Base (PIB) defines the server behavior in all QoS management tasks. This chapter will discuss the design specifics of PIB and the required changes to enable inter-service borrowing. The chapter starts by first describing the overall design of a SAAM server. Then the PIB and its main functions are described in detail. Finally, we conclude with the design details of inter-service borrowing and PIB re-design in support of the new QoS management capabilities of SAAM.

A. THE SAAM SERVER

1. Overview

The basic component of the SAAM prototype is the SAAM router. In its simplest form, the SAAM router is a Java based application that uses a layered architecture to emulate the full Internet Protocol stack and all the functionality associated with a SAAM router. Java multithreading is highly used, which allows all different emulated router components, to function in parallel and to establish data and control channels among them. Although the router model is designed to work over current IPv4 networks, the current version of the SAAM prototype supports only IPv6. The emulated physical layer performs the bridge interface between the underlying IPv4 infrastructure and the IPv6 SAAM prototype.

The SAAM server is an extension of a SAAM router. The server application resides in the outer layer of a SAAM router, i.e. the application layer of the protocol stack. That router is then able to perform all the normal functions associated with a router plus all tasks specific to the SAAM server. Figure 4.1 depicts this modularized approach.

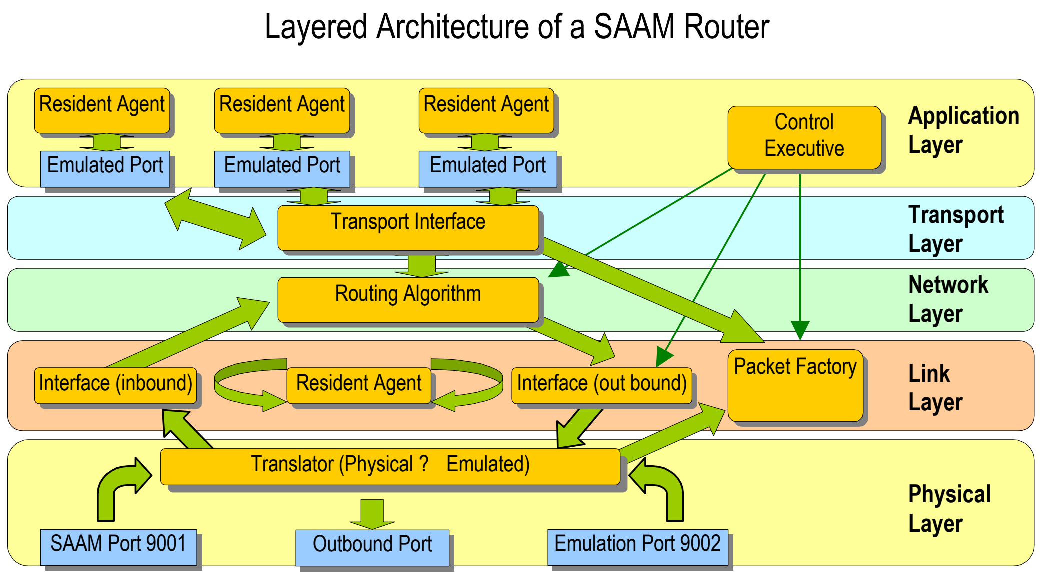


Figure 4.1 Layered architecture of a SAAM router. The SAAM server is a SAAM router with a resident agent application that performs server specific operations.

2. Main Functions of the SAAM Server

The server model was originally developed by NPS graduates Vrable and Yarger [7]. Since then, several enhancements have been made, namely the introduction of the basic QoS management capability by Queck [3], signaling channels by Akkoc [8], backup server functionality by Kati [9], and dynamic PIB generation by Cheng and Gibson [2]. The functions of the current SAAM server prototype may be divided into three types as described in the sections that follow.

a. Active Network Control

When the server first initiates, it tries to establish communication with other SAAM players (SAAM routers and eventual SAAM backup servers). This is done during the initial configuration cycle, by sending special control messages through all server interfaces. Routers in turn will receive this message and continue flooding the rest of the network until edge routers are reached. This initial configuration cycle closes with messages being exchanged among routers and back to the server. When the initial cycle is complete, all participating routers will have registered the presence of a server in the region and updated their routing table to create signaling channels for forwarding control traffic from and to the server [8]. Active network control involves also security related tasks like authentication and integrity protection of control traffic [9].

b. Topology Management

After the initial startup phase and when all signaling channels have been established, participating routers will then take the initiative of reporting their capacities to the server by sending Link State Advertisement (LSA) messages. The first LSA sent by a router typically describes the physical characteristics of each of its interfaces, including the IPv6 address and associated link bandwidth. As the server processes LSA messages from routers, it creates an image of the network by aggregating all pairs of connected interfaces into point-to-point links and discovering all possible path combinations from these links. During normal operation, the topology may dynamically change as interfaces are added or removed through LSA messages.

c. QoS Resource Management

Resource management is the main and most work-intensive function of a SAAM server. It refers to those server actions directly related with maintaining the performance status of all links in its region, the resource reservation and user admission control.

B. THE PATH INFORMATION BASE

As the central repository of information about all paths connecting pairs of routers in the SAAM region, the Path Information Base (PIB) is the core component of the SAAM server and is one of the most important modules of the SAAM architecture. The basic design details of PIB have been documented in [2]. The PIB exchanges information with the network by means of four SAAM specific messages. Two of these messages, the link status advertisement (LSA) and the flow request messages are inbound messages and forwarded by the server agent to the PIB module. In response to each flow request message of a client application, the PIB sends a flow response message to the client and under some conditions, a routing update message to the relevant routers. Both messages are generated within the PIB module and sent to the hosting server, which in turn forwards them to the destination routers. Figure 4.2 shows the basic input/output of PIB as described above.

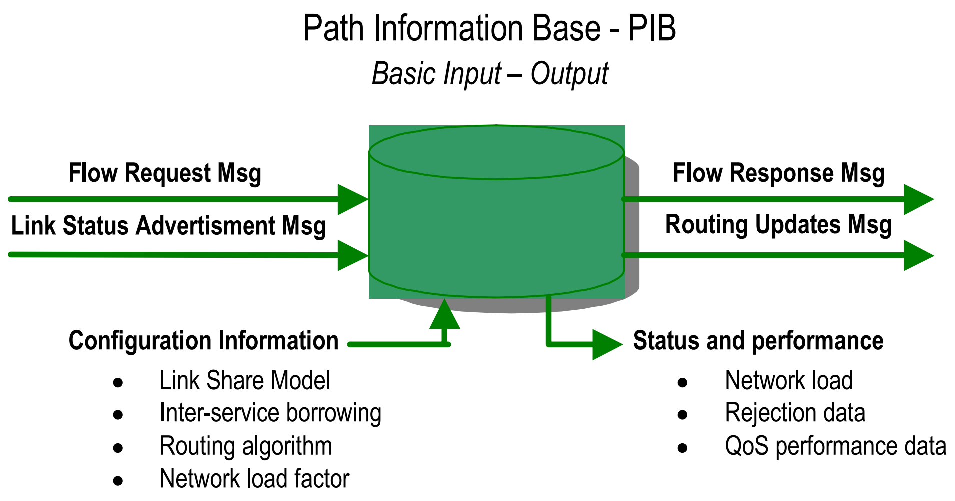


Figure 4.2 The basic input/output channels in the Path Information Base

In addition to the message exchanges during normal network operations, the PIB may accept configuration information during initial startup and, report network status and performance data to an external module. That external module could be part of a network management tool that is capable of sending set or query commands to the PIB, either remotely through especial SAAM messages or locally by conventional console access. An example of such commands would be to turn inter-service borrowing on or off. The development of such functionality is not part of this study and is left for future work.

Figure 4.3 shows the different functional blocks within PIB and the interactions among them. Those functional modules may are organized about two different functional goals: topology management and resource management.

1. Topology Management

Topology management refers to the ability of PIB to discover the topology of the SAAM region and maintain status information about all links. As shown in the top block of figure 4.3, it is achieved by processing LSA messages that the PIB receives from supported routers. Each LSA contains smaller chucks of information called Interface State Advertisement (ISA), each one associated with one of the interfaces of the reporting router. A single ISA may be of typeADD, REMOVE or UPDATE, in order for the PIB to respectively add a new interface, remove an existing interface or update the interface status. The interface status data maintained by PIB are those that are relevant for providing QoS guarantees as defined in SAAM, i.e. the physical link bandwidth, and the link bandwidth utilization, average queuing delay and packet loss rate per service level as observed and advertised by the hosting router.

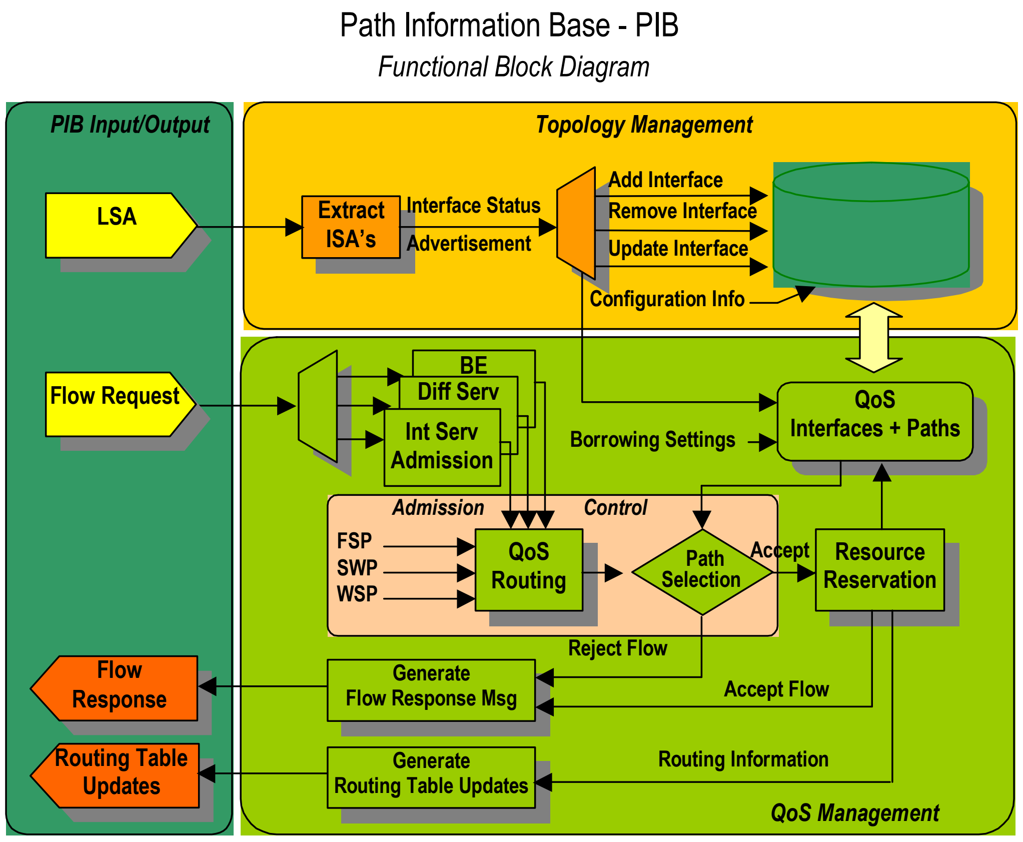


Figure 4.3 The main functional blocks of PIB.

As the PIB adds or removes interfaces, it dynamically regenerates the topology associated with its SAAM region and rearranges the collection of possible paths interconnecting supported routers.

2. QoS Management

When an interface is first advertised, the PIB records its hosting router id, the physical bandwidth, its IPv6 based identification and the information about the neighbor interface, as derived from both the neighbor router id and the number of bits of the subnet mask. The total interface bandwidth is then partitioned into smaller portions and assigned to individual service levels in accordance with the SAAM service model. The current

SAAM prototype supports five different service levels - IntServ, DiffServ, Best Effort, Out of Profile and SAAM control channel.

a. Interface Information in PIB

As already stated, each interface in PIB maintains information about observed QoS per service level (e.g. observed utilization, delay and loss rate), which are periodically refreshed by LSAs containing ISAs of type UPDATE. Based upon the QoS data and the allocated bandwidth per service level, the PIB also computes and stores the available bandwidth per service level. Since the available bandwidth depends on the utilization level as reported by routers, it should be recalculated every time the utilization level changes for each of the service level.

b. Path Information in PIB

As new interfaces are added to PIB, new paths are also created. Each path element in the PIB contains a unique 16-bit integer based path id, the sequence of outbound interfaces the path traverses, and the path's QoS properties in terms of maximum available bandwidth, the bound on end-to-end packet delay and packet loss rate. Each path QoS property can be expressed as a function of the same QoS property of each interface traversed by the path. While the packet delay bound and the loss rate of the path are the summation of the delay bounds and loss rates of the interfaces respectively, the path available bandwidth is the minimum available bandwidth among all interfaces, i.e. the available bandwidth of the bottleneck interface.

c. Processing a Flow Request

When processing a new flow request, the first step, which is common to all requests, is to ensure that source and destination nodes are physically connected, i.e., there is a path in PIB connecting the two nodes. If the destination is unreachable from the source, then the client application will be notified via a flow response message. Otherwise, depending upon the type of request, the admission sequence follows a different procedure as described below.

The simplest admission sequence applies for best effort flow requests. In this case, among all paths available from the source to destination, the admission control module selects one of them in accordance with the predefined path selection scheme. For best effort, no resources need to be reserved, however, if the path has not yet been set up, the resource reservation module ensures the routing tables at each router the path traverses are updated before the affirmative flow response is sent back to the requestor.

The admission sequence for IntServ and DiffServ are similar with the exception that DiffServ requires the additional identification of the network customer running the client application and the confirmation of his Service Level Agreement with the network [3]. The specifics of the agreement and thus the design of this admission step depend upon the service model supported. For both IntServ and DiffServ the next step will then be the selection of a path that meets the QoS requirements of the new flow. At this stage, the admission control unit of PIB simply compares the QoS requirements of this new flow with the QoS properties of all feasible paths to determine the most suitable path. The searching strategy is dictated by the selected routing scheme (i.e., the specific criteria for determining the most suitable path) and by the eventual need to perform inter-service borrowing. Once the path is selected, the resource reservation module of PIB then ensures that the PIB state is updated appropriately to reflect the fact a set of interfaces have just set aside a portion of their resources to support the new flow. This update process directly involves changes to the QoS properties of the path to which the flow is admitted and potentially, of other paths that traverse common outbound interfaces. Furthermore, when a flow is admitted through inter-service borrowing, the update process must be duplicated across the two service levels involved. Finally, a flow response message is sent to the client application. The response is either a flow approval containing the designated flow label that the client application must use to mark its packets, or a flow rejection indicating that no path can meet the QoS requirements of the flow request.

d. Path Selection and QoS Routing

Path selection refers to choosing the best path among all feasible paths that are able to support the QoS parameters as specified in the flow request. There are different approaches to QoS routing [3]: Widest-Shortest Path (WSP), Shortest-Widest Path (SWP) and Shortest-Distance Path (SDP). SWP emphasizes on preserving network resources by selecting a shortest path while WSP provides for load balancing by first selecting widest paths. Due to time constraints, the analysis of the most appropriate schema is left for future research. For the purpose of this thesis, the selected approach is a simplified version of the SWP to what we call First Shortest Path (FSP). The current PIB stores paths with the same source and destination nodes in one array sorted in the increasing order of their hop counts. FSP yields the best possible searching time by retrieving the first path in the array that meets the QoS requirements of the flow request. When inter-service borrowing is enabled and regardless of the route selection schema, inter-service borrowing should only be considered after all paths in the array have been evaluated first without considering inter-service borrowing.

C. INTER-SERVICE BORROWING DESIGN

The underlying theory for inter-service borrowing was already discussed in the previous chapter. Inter-service borrowing only has to do with resource management thus all changes made to the SAAM prototype to enable inter-service borrowing were confined to the PIB module. The following sections describe the design specifics of inter-service borrowing.

1. Inter-service Borrowing at the Interface Level

It was previously stated that PIB maintains two types of QoS data per service level: observed QoS as reported by routers, and available bandwidth as calculated within the PIB. Figure 4.4 below illustrates the typical bandwidth partitioning within one of the service levels participating in inter-service borrowing (DiffServ in this example).

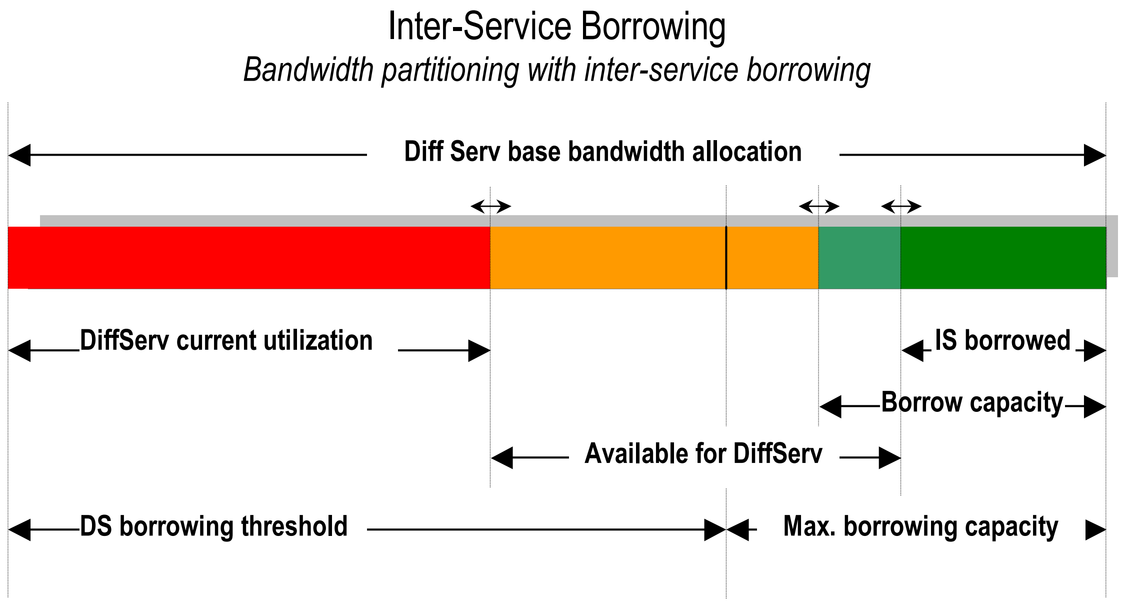


Figure 4.4 Bandwidth partitioning within a single service level that participates in inter-service borrowing (DiffServ in this example).

The whole bar represents the bandwidth base allocation initially assigned to DiffServ. The current DiffServ utilization is shown in red on the left, indicating that the service is under-utilizing its allocated bandwidth, thus suggesting that some capacity could be made available for inter-service borrowing. On the right side of the bar, it is observed that IntServ is already borrowing some capacity from DiffServ (dark green). The capacity made available for IntServ is calculated within PIB as a function of the DiffServ current utilization and the borrowing threshold, i.e., the maximum capacity that might be made available for borrowing. During the admission control sequence, only some of these quantities are of interest. For instance, for admitting a DiffServ flow through this interface, it suffices to know the bandwidth availability of DiffServ. When performing the admission control for IntServ, it only matters to know the borrowing capacity of DiffServ and the amount previously borrowed. With these assumptions and considering the other two QoS metrics of interest (packet delay and loss rate), the following are the minimum set of data associated with a single service level of each interface within PIB:

- Current bandwidth utilization (reported by routers with LSAs)
- Packet queueing delay (reported by routers with LSAs)

- Packet loss rate (reported by routers with LSAs)
- Available bandwidth (calculated in PIB)
- Borrowing capacity (calculated in PIB)

2. Inter-service Borrowing at the Path Level

The path object within PIB contains only those elements of information that are relevant to the admission control function. For the path selection process, it suffices to know the available bandwidth, with and without inter-service borrowing, the packet delay from origin to destination and the packet loss rate for each of the supported service levels. This information is computed from available data stored at each of the interfaces traversed by the path. Figure 4.5 illustrates how the information about three interfaces is used to produce the available bandwidth for a path traversing those interfaces.

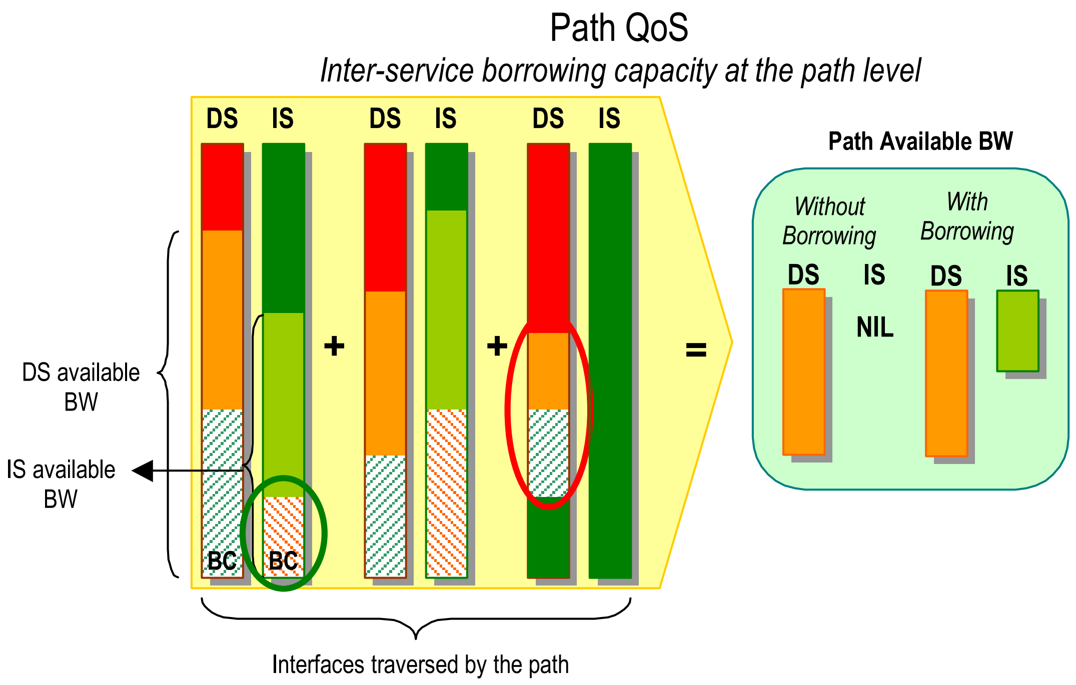


Figure 4.5 Obtaining path QoS information from the interface data

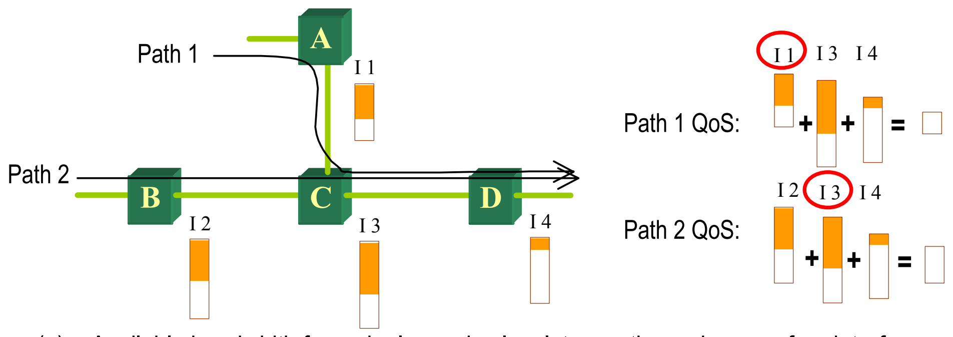
In the example, the third interface shows a high utilization level of IntServ, which is already borrowing some capacity from DiffServ (dark green). Consequently, this path cannot admit any more IntServ flows unless inter-service borrowing is considered. This is shown in the right box, which contains the information associated with the path as calculated from the interface status. Despite not being shown in the figure, in addition to the available bandwidth, the path object contains also the end-to-end packet delay and packet loss rate information.

3. Propagating Interface State Changes

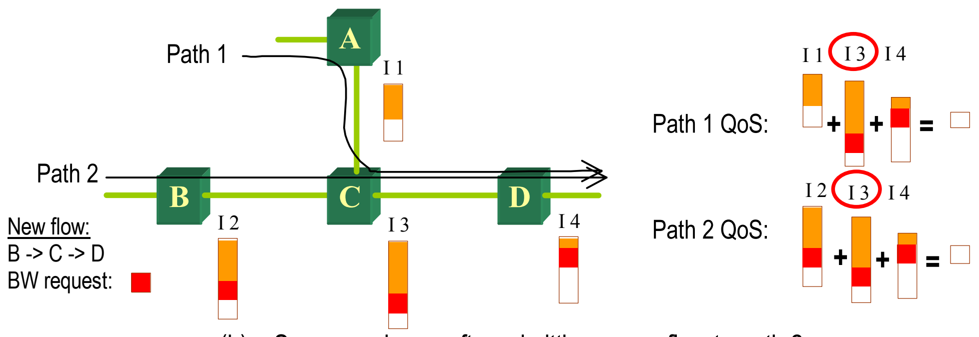
The QoS properties of a path may change due to the admission of a new flow or due to changes of observed performance in one or more of the interfaces it traverses. After processing a LSA of type UPDATE, one or more interfaces in one more service levels may have their observed QoS data modified. In such case, the set of paths that traverse the modified interfaces must also be evaluated and eventually changed to reflect the new state of the interfaces. As already mentioned, the LSA update mechanism allows the PIB to follow a soft state approach in keeping track of admitted flows. Once flows are admitted, there is no need for an explicit flow termination notification and it suffices for PIB to maintain per-interface and per-service level actual aggregated throughput. While changes in service level bandwidth utilization may have no impact in the path information, path delay and loss rate changes will immediately affect the QoS properties of all paths traversing the interface.

A second reason for an interface to change its state is the admission of a new flow. Figure 4.6 shows an example of two paths that share two interfaces in routers C and D. In 4.6(a), it can be observed that interfaces 1 and 3 are the bottleneck for paths 1 and 2 respectively. After admission of a new flow to path 2, all interfaces along the path have their available bandwidth reduced by the amount that is allocated to the new flow. The following step is to propagate changes to those interfaces to all other paths traversing them. In this case, it is observed that the QoS information of path 1 also changes. Moreover, the bottleneck router for path 1 changed from router A to router C (from interface 1 to interface 3).

Interface and Path QoS



(a) – Available bandwidth for a single service level, two paths and across four interfaces.



(b) – Same as above, after admitting a new flow to path 2.

Figure 4.6 Propagation of interface state changes after admission of anew flow

The example of figure 4.6 is a rather simple illustration of the propagation of interface state changes. Utilization changes in any of the two services participating in inter-service borrowing are likely to propagate not only across the QoS of paths and interfaces within the service level of the flow request, but also from one service level to the other. For instance, if a new IntServ flow is admitted using capacity borrowed from DiffServ, changes will occur in the available bandwidth of both services. The propagation of such changes is likely to become process-intensive, especially in large SAAM regions and with a high rate of flow request arrivals.

4. Performance Issues

The complexity and the associated performance concern outlined in the previous section, was always present during the redesign of PIB. While the implementation of inter-service borrowing represents an overhead for the PIB computation, some aspects of the previous PIB design have been modified, which are thought to represent an improvement over the overall PIB efficiency. For instance, when processing an ISA, the interface data only changes if the reported data vary above a predefined variation threshold. In any case, eventual changes will only be propagated after all service state advertisement (SSA) information blocks are processed for that interface. Initially, the propagation of changes occurred after processing each of the SSAs, which could happen as many as fifteen times per each ISA, i.e. the number of service levels (five) multiplied by the number metrics (three – utilization, delay and data loss rate). The revised PIB ensures that the propagation of changes happens at most once per ISA processed, which represents a considerable efficiency improvement over the previous version.

During normal network operation, PIB will be processing either link status messages or flow requests. The arrival rate and the size of link status messages are dictated by the predefined LSA cycle duration and the number of router and interfaces supported in the SAAM region. The overhead represented by the arrival of flow requests depends upon the amount of network resources, e.g. link capacity, and other external factors like average flow request rate and flow duration. While maintaining a short LSA cycle is desirable for an increased accuracy of PIB to better respond to new flow demands, it also represents an increase in processing overhead. Such overhead may be excessive in presence of a high rate of flow requests. Long LSA cycles however, shall be avoided especially is the network load changes very rapidly. Because of time limitations, choosing optimum values for those parameters is outside the scope of this study and is left for future evaluation.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SAAM RESOURCE MANAGEMENT IMPLEMENTATION

The SAAM prototype has been developed over the past three years by several students contributing to the SAAM project with their thesis efforts. As part of this thesis study, the new resource management concept as described in the previous chapters was fully implemented in Java and integrated into the Java based SAAM prototype. The implementation and integration details will be described in this chapter. Appendix B contains the SAAM source code that was either created or modified to incorporate the new functionality. With the embodiment of this functionality, the SAAM prototype now has a greatly improved resource management capability.

A. OVERVIEW

As described in chapter IV, the Path Information Based (PIB) is the core element of a SAAM server. The PIB module is self-sufficient, containing both network data and operations that are responsible for the SAAM server behavior. SAAM resource management behavior is therefore fully provided by PIB. Consequently, the PIB was the only module of SAAM that was modified to incorporate the new resource management mechanism detailed in this thesis. The PIB is implemented by the `BasePIB` java class, which in turn contains several inner classes. Table 5.1 contains the complete listing of all those classes with a brief description about the purpose of each one. The detailed description of each of the `BasePIB` data members is fully documented in [2]. Thus, the focus of this chapter is on the parts of PIB that were created or modified as part of this thesis.

The implementation of inter-service borrowing required an extension of the existing data structures that supported SAAM and a redesign of the PIB behavior implementation. Most of the changes directly related with data objects, such as the `InterfaceInfo` and `Path` objects, were accomplished by incorporating additional data members. This was required in order for those objects to keep track of inter-service borrowing capacity, and path available bandwidth. The redesign of the PIB behavior was achieved through a number of changes to existing `BasePIB` methods and additions of several new methods. In addition, the path selection mechanism was tightly integrated into the existing `BasePIB` code. A new class named `RoutingAlgorithm` was created to encapsulate the path selection behavior, making it modular and ready for further development with regard to the route selection process.

Class Name	Description
Main PIB class:	
*saam.server.**BasePIB***	Main class that implements the PIB
PIB inner classes:	
*Saam.server.BasePIB.**InterfaceInfo***	For the interface object.
*saam.server.BasePIB.**ObsQoS***	For the interface observed QoS.
*saam.server.BasePIB.**PathQoS***	For the path QoS information.
*saam.server.BasePIB.**Path***	For the path objects.
*saam.server.BasePIB.**aPIIndex***	The collection of paths in PIB.
*saam.server.BasePIB.**FlowQoS***	For the QoS flow properties.
*saam.server.BasePIB.**RoutingAlgorithm***	For the path selection scheme.

Table 5.1 List of PIB classes

The following sections detail all the changes mentioned above, starting with all the changes made to the inner classes of *BasePIB,* followed by modifications applied to the *BasePIB* methods, and finally, the implementation of the new *RoutingAlgorithm* class.

B. IMPLEMENTATION DETAILS

This section briefly describes *BasePIB* and each of its inner classes. Each of the sub-sections describes a single class, starting with a general description of the class, and followed by a table of the class's data members and its most important methods.

1. BasePIB Support Classes

Several support classes have been implemented as inner classes of the main `BasePIB` class. This approach can be justified because their functions are only relevant for the PIB itself, which is therefore self-contained within the `BasePIB` class. The following sections briefly describe each of those classes.

a. InterfaceInfo Class

The `InterfaceInfo` class defines the object that contains all key information required by PIB to describe a single SAAM router interface. This object contains static and dynamic information about the interface. The static information is related with the network configuration and is loaded upon initial interface advertisement (ISA-ADD). The dynamic information changes very often upon processing LSA or flow request messages. `InterfaceInfo` data members are listed in the table below.

Name	Type	Description
aObjObjectQoS	*ObsQoS[]*	Array of *ObsQoS* objects indexed by the service level. Maintain status information about observed QoS.
bSubnetMask	*byte*	The number of bits of the subnet mask.
htPathIDs	*Hashtable*	Table with all path paths traversing this interface (outbound direction).
iNodeID	*Integer*	The node id of the hosting router.
iServiceLevelAvailable Bandwidth	*int[]*	Available bandwidth per service level.
iServiceLevelBorrowing Capacity	*int[]*	Borrowing capacity per service level.
ITotalBandwidth	*int*	Physical interface

Name	Type	Description
		bandwidth.
iUnallocatedBandwdith	*int*	Absolute bandwidth not allocated to any service level.

Table 5.2 Data members of `InterfaceInfo` class

The table below lists some of the methods provided by the `InterfaceInfo` class.

Name	Return	Description
getServiceLevelAvailableBandwidth()	*int[]*	Retrieves the available bandwidth for all of the service levels.
getServiceLevelBorrowingCapacity(byte)	*int*	Retrieves the borrowing capacity for the given service level.
getUnallocatedBandwidth()	*int*	Gets the unallocated bandwidth.
resetQoS()	*void*	Resets the QoS array.
setServiceLevelBorrowingCapacity(int)	*void*	Sets the borrowing capacity for a given service level.
setServiceLevelUnallocatedBandwidth(int)	*void*	Sets the unallocated bandwidth.

Table 5.3 Methods of `InterfaceInfo` class

b. ObsQoS Class

The `ObsQoS` class defines an object that is associated with a single service level at each interface. This object contains observed QoS information for that service level as reported by routers through LSA messages. The table below lists `ObsQoS` data members.

Name	Type	Description
iDelay	*short*	The data delay observed at this interface for a given service level.
iLossRate	*short*	The data loss rate at this interface for a given service level.
iUtilization	*short*	The bandwidth utilization of a given service level, as a scaled percentage of the total interface bandwidth.

Table 5.4 Data members of `ObsQoS` class

c. PathQoS Class

The `PathQoS` class defines an object that is associated with a single path. This object contains relevant end-to-end QoS and resource availability information for a particular service level. The table below lists `PathQoS` data members.

Name	Type	Description
PathAvBW	*int*	The path available bandwidth for a given service level.
pathAvBWwBorrowing	*int*	The path available bandwidth for a single service level, possible incremented by borrowed bandwidth.
PathDelay	*short*	The total path delay for a given service level, i.e. current delay experienced by traffic along the path.
pathLossRate	*short*	The current total data loss rate for a given service level, experienced by traffic along the path.

Table 5.5 Data members of `PathQoS` class

Method Name	Return Type	Description
getAvailableBandwidth()	*int*	Retrieves path available bandwidth for the service level.

Method Name	Return Type	Description
getAvailableBandiwdthIncludingBorrowing()	*int*	Retrieves path available bandwidth for the service level and considering borrowing.
setAvailableBandwidth (int)	*void*	Sets a new value for available bandwidth.
setAvailableBandwidthIncludingBorrowing(int)	*void*	Sets a new value for available bandwidth including borrowing

Table 5.6 Methods of *PathQoS* class

d. Path Class

The *Path* class defines an object that is associated with a single path. The table below lists data members of the *Path* class.

Name	Type	Description
ahtFlows	*Hashtable[]*	An array of look-up tables for flows assigned to a path. Array is index by service level.
bCreated	*boolean*	Signals whether routing tables have been update at the routers traversed by this path.
iNewFlowID	*int*	For the assignment of flow IDs.
iPathID	*Integer*	The ID of this path.
mirrorPath	*Path*	The mirror path, i.e. the path that traversed the same interfaces but in opposite direction.
objaPIIndex	*aPIIndex*	A cross-reference to the array of path objects.
objPathQoS	*PathQoS[]*	The array of *PathQoS* objects that contain the QoS information of this path, per service level.
vInterfaceSequence	*Vector*	The *Vector* object containing the sequence of interfaces traversed by this path (outbound), listed from destination to source node.

Name	Type	Description
vNodeSequence	*short*	A `Vector` object containing the sequence of nodes (routers) the path traverses, listed from destination to source.

Table 5.7 Data members of `Path` class

e. aPIIndex Class

This class is a data member of the `Path` class. It is used to create an index object for quick access to the path information array `aPI`. The index fields are source node, destination node and hop count. Given a path object, this object can be used to obtain the `aPI` element (a vector of paths) that contains the path.

Name	Type	Description
iDestination	*Integer*	The last node ID of the path.
iHopCount	*int*	The hop count of the path.
iSource	*Integer*	The first node ID of the path.

Table 5.8 Data members of `aPPIndex` class

f. FlowQoS Class

The `FlowQoS` class defines an object that characterizes the QoS of a single flow request. The table below lists `FlowQoS` data members.

Name	Type	Description
requestedBandwidth	*int*	The bandwidth capacity being requested.
requestedDelay	*short*	The requested delay bound.
requestedLossRate	*short*	The requested loss rate bound.
timeStamp	*long*	When the flow request is made.

Table 5.9 Data members of `FlowQoS` class

g. *RoutingAlgorithm Class*

The `RoutingAlgorithm` class implements a stateless object. Routing algorithm objects behave like function objects, providing only the path selection functionality to PIB, based on the selected routing algorithm. The current version of this class only implements the First-Shortest Path (FSP) algorithm but the class can be easily extended to provide different path selection schemes.

Method Name	Return Type	Description
findPath(src, dest, algorithm)	*Path*	Selects a path from PIB, between src and dest, using the selected algorithm.
findPath(src,dest, qos,sl,borrowing, algorithm)	*Path*	Selects a path from PIB, between src and dest, using the selected algorithm and meeting the required QoS demand with borrowing as option.

Table 5.10 Methods of `RoutingAlgorithm` class

2. BasePIB Class

The `BasePIB` class is the main PIB class. Within the SAAM server, the PIB object is an instance of the `BasePIB` class, extending the underlying organization as inherited from the `PathInformationBase` abstract class. The `BasePIB` comprises several data members organized to create a complex data structure, which ultimately represents an image of the SAAM region, containing both static and dynamic information about the supported SAAM network. As stated before the full detail about the PIB implementation can be found in [2]. For the purpose of this thesis, it is only relevant to cover the parts of PIB that have been modified or created for this thesis or are dimmed to be important for describing the implementation of the new resource management and inter-service borrowing mechanisms. The table below lists the most relevant `BasePIB` data members.

Name	Type	Description
afBASE_ALLOCATION	*float[]*	The fraction of the total interface bandwidth, initially

Name	Type	Description
		allocated to each service level.
afLOAD_FACTOR	*float[]*	The maximum load capacity within each of the service levels.
aPI	*Hashtable[][][]*	A tri-dimensional array of hash tables containing all paths between any nodes for a given hop counts.
BORROWING_PROB_OFFSET	*float*	The borrowing limit as an offset factor, depending upon the selected probability (95% equates to 0.164).
BORROWING_THRESHOLD	*float*	The borrowing threshold.
DISPLAY_FULL_DETAIL	*boolean*	Whether the PIB gui will display full detail.
htInterfaces	*Hashtable*	Collection of *InterfaceInfo* objects, keyed by Interface address string.
htPaths	*Hashtable*	Collection of all *Path* objects, keyed by path id integer object.
routingAlgorithm	*RoutingAlgorithm*	The instance of the Routing Algorithm.

Table 5.11 Data members of *BasePIB* class

The table below lists some of the relevant *BasePIB* methods

Method Name	Return Type	Description
admissionControl_BE (FlowRequest)	*int*	Performs the admission sequence for BE.
admissionControl_DS FlowRequest()	*Int*	Performs the admission sequence for DiffServ.
admissionControl_IS	*int*	Performs the admission

Method Name	Return Type	Description
(FlowRequest)		sequence for IntServ.
BorrowingCapacity (bw, utiliz,sl)	*int*	Calculates the borrowing capacity given the service level bandwidth, current utilization and the service level.
FindPathDelayAndLossRate (Path)	*short[][]*	Discovers path delay and loss rate.
isRouteFeasable(src,dest)	*boolean*	Checks if a path exists between *src* and *dest*.
pathBandwidth(Path)	*int[][]*	Discovers path available bandwidth, with and without borrowing, for all service levels.
pathBandwidth(Path, sl)	*int[]*	Discovers path available bandwidth, with and without borrowing, for the given service level.
ProcessFlowRequest (FlowRequest)	*int*	Processes a new flow request.
ProcessLSA (LinkStatusAdvertisement)	*void*	Processes incoming LSA messages.
RefreshInterfaceBW (InterfaceInfo, bandwidth reduction, svc level)	*void*	Refreshes the interface available bandwidth following the admission of a new flow.
RefreshInterfaceQoS (InterfaceInfo)	*void*	Refreshes the interface available bandwidth after processing an LSA that incurred changes.
refreshPathQoS(Path)	*void*	Refreshes path QoS.
RefreshPathQoS (Path, qos variations)	*void*	Refreshes path QoS based on the provided QoS changes.
resetQoS()	*void*	Resets PIB QoS information .

Method Name	Return Type	Description
setInterserviceBorrowing(Boolean)	*void*	Sets the state of inter-service borrowing.
setupPath(Path, int)	*void*	Sets up a path, by sending the routing table update messages to routers.
updateAvailableBandwidth(Path, bandwidth, svc level)	*void*	Updates the available bandwidth of a path after a flow admission.
updateInterface(InterfaceSA)	*void*	Processes ISAs for updating an interface with observed QoS

Table 5.12 Methods of *BasePIB* class

C. MAJOR PIB MODIFICATIONS

The main functions of PIB were outlined in chapter IV. The previous section briefly listed main components of the PIB implementation. In this section, the PIB operation is described in terms of the two main PIB functions, i.e., processing LSA and Flow request messages.

1. LSA Message Processing

Whenever an LSA is to be processed by PIB, the parent object of PIB (Server) invokes *processLSA()*. This method first checks if it the LSA is from a new node. In that case, the new node is created and added to PIB. In either case, the LSA is stripped into one or more ISAs. Finally, depending upon the type of ISA, *updateInterface(), removeInterface()* or *addInterface()* methods is invoked once per ISA.

a. *addInterface()*

When an interface is first advertised by a router, the respective *InterfaceInfo* object is created and stored in PIB. The *InterfaceInfo* constructor initializes interface data and then calls the *refreshInterfaceQoS()* method.

b. updateInterface()

This method is invoked from `processLSA()`, to process a single ISA-UPDATE. An ISA may in turn contain one or more SSAs for that interface. Each metric value of each SSA is extracted and the respective variable in the interface QoS array updated if the changes is above a predefined threshold. The two arrays `deltaLossRate` and `deltaDelay` keep track of all changes introduced while processing all SSAs in this ISA. After all SSAs are processed, then if the observed QoS of this interface effectively has changed, `refreshInterfaceQoS()` is called. Finally, when the QoS properties of the interface has changed, for every path traversing this interface, the `refreshPathQoS()` is called, providing as arguments the `deltaDelay` and `deltaLossRate` arrays.

c. refreshInterfaceQoS()

This method may be invoked by the `InterfaceInfo` constructor during initial interface object instantiation or from the `updateInterface()` method, after processing all SSAs of a single ISA. In either case, it first calculates the unallocated bandwidth from the current service level utilizations and total interface bandwidth. For each service level, it calculates the available bandwidth, service level base allocation and borrowing capacity, all as a function of current utilization and base allocation. For this last purpose, the method `borrowingCapacity()` is invoked.

d. refreshPathQoS()

The purpose of this method is to refresh the path QoS after changing the QoS properties of one outbound interface along that path. When invoked from the `updateInterface()` method, the provided `deltaDelay` and `deltaLossRate` information allows rapidly setting of the new path delay and loss rate values. However, for determining the path available bandwidth, it is still required to visit all interfaces traversed by the path. This is achieved by invoking the method `pathBandwidth()`. An overloaded version of this method is invoked by any of the four path constructors, for the initialization of the path QoS. In this case, the path delay and data loss rate is obtained by invoking the method `findPathDelayAndLossRate()`.

e. borrowingCapacity()

Given the current utilization, the base allocation and with the predefined offset factor and borrowing threshold, this method calculates the borrowing capacity of a service level.

f. pathBandwidth()

The method traverses all interfaces of a given path to discover the minimum available bandwidth per service level. For each service level, two values are obtained: available bandwidth with and without inter-service borrowing.

g. findPathDelayAndLossRate()

This method traverses all interfaces of a given path. For each of the service levels, it calculates the path delay and data loss rate across the entire path.

2. Flow Request Message Processing

A Flow Request message is passed to PIB by the Server object invoking the method `processFlowRequest()`. This is the starting point for the admission control sequence, which follows different paths depending upon the service level of the flow request. The following is the description of all the methods that are directly involved in this admission control sequence.

a. processFlowRequest()

This method simply receives the Flow Request and then, depending upon the service level of the request, invokes the respective admission control method.

b. admissionControl_BE()

The admission sequence for a BE is the simplest. The information about source and destination routers is first extracted from the message. Then the routing algorithm object is called to select the path and finally the flow response is generated with the flow label information and sent back to the source router

c. admissionControl_IS()

The admission sequence for IntServ starts by first extracting the request information including the QoS parameters. The method `isRouteFeasable()` is called to verify if a valid path exists, connecting source and destination routers. If that path exists then the routing algorithm object is invoked to select the best path according to the requested QoS parameters and the selected routing algorithm. Then if it succeeds, network resources will be allocated by calling the method `updateAvailableBandwidht()`. Finally, the new flow is added to the selected path, a Flow Response message is generated and sent to the client application and, if necessary, the routing table updates for the path are sent to the routers across the path.

d. admissionControl_DS()

The admission sequence for a DiffServ flow request differs from the IntServ sequence only in the fact that requesting users need to be identified and their QoS parameters are extracted from the pre-established service level agreement with those users.

e. isRouteFeasable()

This method simply searches the array of path information (`aPI`) for ensuring that there exists at least one physical path between source and destination.

f. updateAvailableBandwidth()

This method performs the resource reservation step in the admission sequence. The arguments are the path, the granted bandwidth and the target service level. Adding a flow to a path reduces the available bandwidth of all interfaces traversed by that path. This method ensures that all of those interfaces are visited and for each one, the method `refreshInterfaceBW()` called to perform the respective interface QoS update.

g. refreshInterfaceBW()

This method is invoked once for every interface traversed by a given path, after admission of new DiffServ or IntServ flow. The available bandwidth for the target service level is reduced by the same amount granted to the new flow. However, the refresh algorithm needs to determine in which stage the bandwidth allocation is at, i.e. direct, dynamic or inter-service borrowing. This is relevant to calculate the new unallocated bandwidth and borrowing capacity. Eventual changes in the unallocated bandwidth are also reflected in the available bandwidth for the other service level. For instance, if the admission of a new IntServ flow decreases the unallocated bandwidth, final available bandwidth for both IntServ and DiffServ need to be updated. Finally, changes in the available bandwidth of a single interface propagate to paths traversing that interface. Consequently, all paths traversing affected interfaces need to be evaluated for eventual path QoS changes. For that purpose, the method *pathBandwidth()* is called. For efficiency reasons, this last path update phase is not done immediately after a single interface change is introduced. Instead, as interfaces are updated, the refresh algorithm keeps track of the affected paths. At last, when all affected interfaces have been updated, each of those paths is evaluated. The efficiency gain is more evident when more than one interface is traversed by the same path.

D. CONTROLLING PIB BEHAVIOR

It is desirable to control the behavior of PIB as it interacts with the other network players. Some of that control may be static and is therefore hard-coded and other can be used by other SAAM modules, like a future network management console. The following table summarizes those parameters of PIB that may change its configuration and the way it operates.

Name	Description
final boolean *TESTING_MODE*	Defaults to false. When set to true, a PIB Tester class is instantiated. The PIB tester has its own gui and can be used to inject flow requests and LSA messages in PIB as well as other interactions with PIB. Chapter VI covers this tester in detail.
final boolean *DISPLAY_FULL_DETAIL*	Defaults to false. When set to true, the PIB gui in the SAAM demo station displays more detailed information about PIB operation. This is ideally suited for PIB diagnosis or to monitor the execution flow within PIB.
final boolean *INTERSERVICE_BORROWING_DEFAULT*	The default state for inter-service borrowing. It defaults to true, which means it is enabled

Name	Description
	means it is enabled.
final float BORROWING_THRESHOLD	The borrowing threshold is the level above which base allocation bandwidth can be made available for borrowing. The default value is 0.6, i.e. 60% of the base allocation.
final float BORROWING_PROBABILITY_OFFSET	This offset factor is derived from the selected probability for inter-service borrowing, as derived from equation 3.11 in chapter III. Default value is 0.164 as given by the probability of 95%.
final float afBASE_ALLOCATION[]	This array defines the base bandwidth allocation for all service levels supported. The share of each service level should be so that the summation is less than or equal to the unit value.
final float afLOAD_FACTOR[]	This array defines the maximum load to be supported for each of service levels.
setInterseriveBorrowing()	Enables to dynamically enable or disable inter-service borrowing.

Table 5.13 PIB configuration elements

VI. PIB TEST AND PERFORMANCE ANALYSIS

The performance evaluation of the proposed resource management technique is a key aspect for this research study. Some of the concepts developed during the previous chapters have a statistical base. A few assumptions require validation. This chapter describes the approach used for testing and evaluating the new PIB, the design and implementation details of the PIB tester module, the specific experiments carried out, and finally, the analysis of the data collected.

A. APPROACH

After completing the implementation, the code was checked for correctness. Every single section was checked for correct logic implementation in accordance with the resource management algorithm. The execution flow and input/output of the different methods was ensured to be as expected. Once the initial check was complete, a functional check of the whole PIB was conducted. The two main functions being checked were the processing of LSA messages and Flow Request messages. LSA generation is not yet completely functional. Current SAAM prototype produces a limited number of LSAs during the initial configuration cycles. Flow generator agents may be used to generate flow request messages. While this approach was acceptable to check the general correctness of the implementation, it was far less than what is required to test the whole resource management algorithm under normal network operation.

Accurate LSA reporting is essential for PIB to maintain coherent status information about the network. However, the link state monitor in the router prototype is not fully implemented, which causes incorrect link status information to arrive to PIB. Another apparent limitation of the current SAAM prototype has to do with the process of generating flow requests and network traffic. While the current system for launching flow generator agents is adequate for a small number of flows, it does not scale well to several hundreds of flows. Without generation of precise LSA messages, it is not possible to test complete PIB operation. Moreover, testing the new resource management capability of PIB requires complete control over a large number of flows.

B. TEST DESIGN

The performance study carried out in this thesis followed a different approach. The SAAM auto-configuration process is run for only one cycle, at the end of which the target PIB is created based on the LSA messages from LSA monitors of all routers. Afterwards, a computer simulation program called PibTester is used to produce all flow request and LSA messages, circumventing the shortcomings of current traffic generator agents and LSA monitors. The theoretical foundations for this evaluation can be found in [11]. The following sub-sections describe the different steps undertaken.

1. System Definition and Testing Goals

As mentioned before, the new resource management scheme was fully implemented within the PIB. The key component of SAAM that is under study is therefore the PIB of a SAAM server. For that purpose, the test is confined to interactions with the `BasePIB` class, including all of its internal support classes. However, the full SAAM prototype should be up and running as normal, reflecting the normal operating environment. The goals of this test will be to (1) evaluate the correctness of PIB resource management operation under normal load conditions; (2) evaluate the impact of inter-service borrowing in terms of efficient management of network resources. In order to attain these goals, the network must work under extreme load conditions for some service, i.e., at the point where inter-service borrowing is required. Additionally, it should suffice to consider the two service levels with dynamic bandwidth allocation, i.e., IntServ and DiffServ.

2. Defining Services

The system service is identified as network resources allocated to network users. In this study, the focus is on the amount of link capacity the network resource management system is able to allocate to users.

3. Selecting Metrics

In the beginning, the test is limited to validating correct operation of PIB. As Flow Request messages arrive, the PIB should be able to complete the admission procedure and allocate resources accordingly. The accurate LSA generation based on traffic of active flows is a major requirement for the testing. After processing these LSA messages, the PIB should accurately refresh its state to reflect new LSA information. The ability to monitor the PIB state in general and the interface and path status information in particular plays a major role in testing the PIB.

In the second phase of testing, we are interested in the number of flows arriving at PIB versus the number of flows that are rejected, in different service levels. This information should be cross-referenced with the number of active flows, from which it is then possible to calculate the expected link load for each of the service levels.

4. Listing Parameters

Several parameters have been identified to affect the resource management performance. These parameters can be divided into system and workload parameters as follows:

a. Systems Parameters

- LSA generation period;
- Borrowing threshold;
- Borrowing probability;
- Service level Base Allocation;

b. Workload Parameters

- Number of active flows;
- Flow duration / requested bandwidth (distribution);
- Inter-arrival time of flow requests;

5. Factors Under Test

The key factor chosen for the purpose of this test is inter-service borrowing. The focus will be on the response of the PIB resource management with inter-service borrowing enabled as opposed to inter-service borrowing disabled. Additionally, different borrowing threshold values will be used to evaluate the impact on flow rejection.

6. Evaluation Technique

Since the PIB is fully prototyped, it is possible to perform a complete functional test of PIB. The measurement technique will be used to collect performance data from PIB. Simulation will be used to generate flow requests and to LSA messages.

7. Workload

The workload for this test consists of a number of flow requests continuously arriving at PIB, following a predefined network load profile. At the same time, LSA generation ensures accurate link state information is reported to PIB. These LSA messages should emulate the tasks of routers advertising link utilization based on network traffic generated by active flows.

8. Designing the Experiments

Two different topologies will be used for testing as described below.

a. Network A

This is the simplest topology and as depicted in figure 6.1, the network will be made of three nodes: one SAAM server and two SAAM routers. The goal is to test PIB resource management across a single link between two routers. For that purpose, the simulation will generate flow requests for traffic going from A to B. The objective is to obtain utilization and flow rejection data about interface 2..1 at router A, as the network load increases.

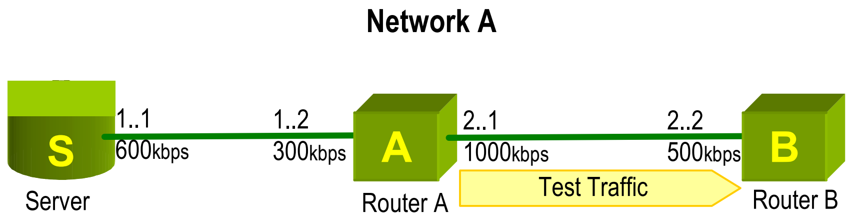


Figure 6.1 Test topology A – 1 server, 2 routers

b. Network B

The focus of network A is to analyze the resource management working at the link level and over a single network link. With network B however, we are interested to observe the behavior of PIB when alternate paths are available and when admission of a single flow indirectly affect other paths. The network is required to be loaded with traffic such that, for any source-destination pair different paths may be selected at different times. Flows are generated from a random source to a random destination among the three routers A, B and C. The selected topology for network B is depicted in figure 6.2.

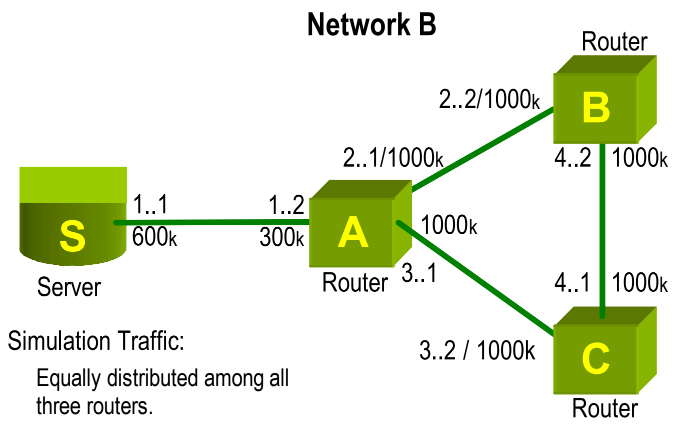


Figure 6.2 Test topology B – 1 server, 3 routers

C. PIB TESTER

1. Tester Requirements

As mentioned in the beginning of this chapter, there were two major factors driving the need for developing a specific PIB tester: accurate LSA generation and effective control over a large number of flow requests. The following basic requirements were identified for that tester:

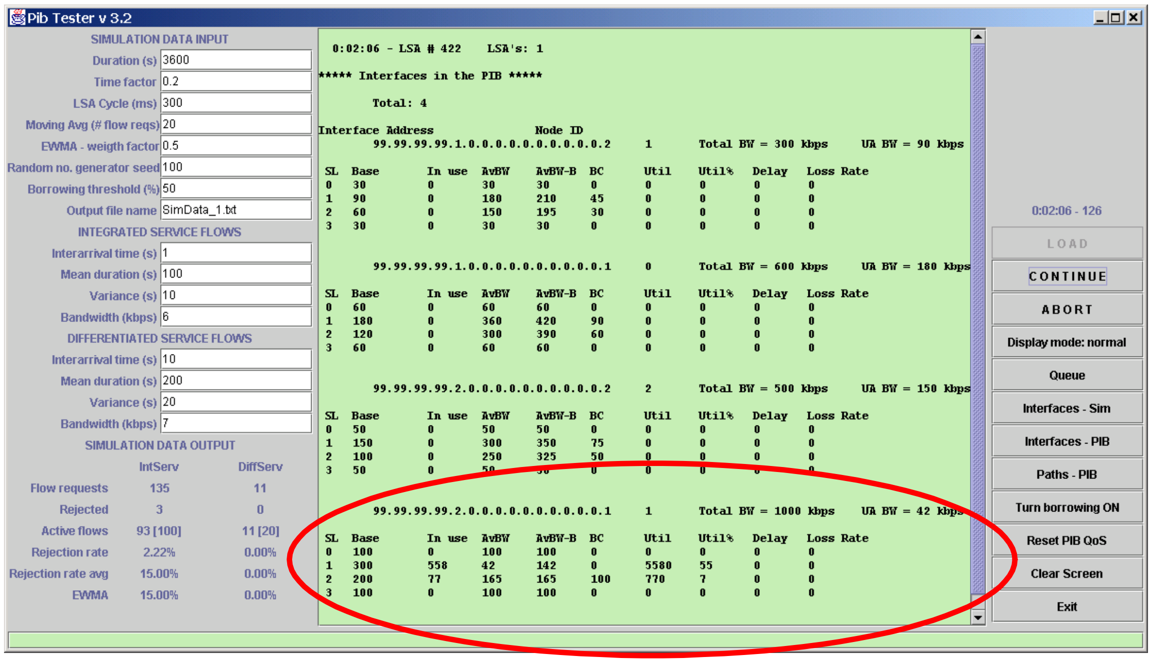
- Generate and send to PIB a stream of flow request messages, in accordance with a predefined link load profile.
- Generate and send periodic LSA messages to PIB. The LSA information should emulate the link status according to the number and characteristics of active flows.
- Store status data every time the system (the PIB and the tester) changes state.
- All testing should be external to PIB.

2. Tester Module

The class `PibTester` was created for the only purpose of testing PIB. `PibTester` is instantiated by `BasePIB` whenever the TESTING_MODE flag in the beginning of `BasePIB` code is set to true. The tester runs in a separated thread and has its own GUI interface. Since the tester is a BasePIB object, it has direct access to BasePIB methods. However, only two methods of BasePIB are used by the tester: `processLSA()` and `processFlowRequests()`. These are exactly the same methods used by the parent class of `BasePIB` (`Server` class) to process incoming LSA and Flow request messages. Synchronization is not required as long as other SAAM players do not send LSA or Flow request messages to the server. The tester automatically waits for initial cycle to complete and no more LSA messages will arrive to PIB from sources other than the tester. Flow generator agents should not be deployed to SAAM routers therefore avoiding flow requests to arrive to PIB. The tester as described above will be the sole player interacting with PIB.

The core element of the `PibTester` is a priority queue, which may contain three types of event objects: flow request, flow termination and LSA. Before a simulation run starts, the priority queue is loaded with flow request and LSA events for the duration of the trial. The number of LSA events is determined by simulation length and the LSA cycle time. The number of flow requests for each of the two services being considered (IntServ and DiffServ) is determined by the respective flow arrival distribution. When a simulation run starts, the simulation time is used to determined when events are dequeued. Every time a flow request event is dequeued, a Flow Request message is sent to PIB. As flows are accepted, the tester updates its own database of interfaces to keep track of active flows and interface utilization. Additionally, a flow termination event is enqueued, with an event time that corresponds to end of the newly admitted flow. When flow termination events are processed, the database of interfaces in the tester is updated to reflect the new network resources becoming available. An LSA event triggers one LSA message for every router represented in the database of interfaces within the tester. The current interface utilization is used to report to PIB the emulated interface status. The LSA will therefore reflect the number of virtual active flows using the network.

During a normal simulation run, the tester GUI displays simulation progress, status information as obtained from the Path Information Base, and limited statistical data. Additionally, at every single simulation event, a line of status data is written to an ASCII file for posterior analysis. Figure 6.3 is a screen shot of the tester interface, during a simulation run of network A. The central area of the display shows individual interface data stored in the Path Information Base. For example, the encircled area contains information about interface 2..1. Appendix 4 contains a detailed description of the tester functionality and appendix 5 contains all of its source code.



SIMULATION DATA INPUT

Duration (s)	3600
Time factor	0.2
LSA Cycle (ms)	300
Moving Avg (# flow reqs)	20
EWMA - weigth factor	0.5
Random no. generator seed	100
Borrowing threshold (%)	50
Output file name	SimData_1.txt

INTEGRATED SERVICE FLOWS

Interarrival time (s)	1
Mean duration (s)	100
Variance (s)	10
Bandwidth (kbps)	6

DIFFERENTIATED SERVICE FLOWS

Interarrival time (s)	10
Mean duration (s)	200
Variance (s)	20
Bandwidth (kbps)	7

SIMULATION DATA OUTPUT

	IntServ	DiffServ
Flow requests	135	11
Rejected	3	0
Active flows	93 [100]	11 [20]
Rejection rate	2.22%	0.00%
Rejection rate avg	15.00%	0.00%
EWMA	15.00%	0.00%

```
 0:02:06 - LSA # 422   LSA's: 1

***** Interfaces in the PIB *****

        Total: 4

Interface Address               Node ID
        99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.0.2     1       Total BW = 300 kbps     UA BW = 90 kbps

 SL  Base       In use  AvBW     AvBW-B  BC       Util     Util%    Delay    Loss Rate
 0   30         0       30       30      0        0        0        0        0
 1   90         0       180      210     45       0        0        0        0
 2   60         0       150      195     30       0        0        0        0
 3   30         0       30       30      0        0        0        0        0


        99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.0.1     0       Total BW = 600 kbps     UA BW = 180 kbps

 SL  Base       In use  AvBW     AvBW-B  BC       Util     Util%    Delay    Loss Rate
 0   60         0       60       60      0        0        0        0        0
 1   180        0       360      420     90       0        0        0        0
 2   120        0       300      390     60       0        0        0        0
 3   60         0       60       60      0        0        0        0        0


        99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.0.2     2       Total BW = 500 kbps     UA BW = 150 kbps

 SL  Base       In use  AvBW     AvBW-B  BC       Util     Util%    Delay    Loss Rate
 0   50         0       50       50      0        0        0        0        0
 1   150        0       300      350     75       0        0        0        0
 2   100        0       250      325     50       0        0        0        0
 3   50         0       50       50      0        0        0        0        0


        99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.0.1     1       Total BW = 1000 kbps    UA BW = 42 kbps

 SL  Base       In use  AvBW     AvBW-B  BC       Util     Util%    Delay    Loss Rate
 0   100        0       100      100     0        0        0        0        0
 1   300        558     42       142     0        5580     55       0        0
 2   200        77      165      165     100      770      7        0        0
 3   100        0       100      100     0        0        0        0        0
```

Figure 6.3 A snap shot of the PIB tester, during a simulation run.

D. DATA ANALYSIS AND INTERPRETATION

Several simulation runs were conducted. Each run produced a single file of raw data, which was then imported into an Excel spreadsheet. Excel was used to separate the data from each of the three types of events and generate time graphs showing the progress of the relevant variables. At this stage, the initial and final transient periods were easily identified. Since we are only interested on the steady-state performance, data collected over those two periods were deleted. From remaining data, it was then possible to obtain two types of information: a graphic representation of variables plotted against time and calculated average values during that sample interval. The following sections present the data as described.

1. Borrowing Sensitivity Test With Network A

For the purpose of this test, the interface 2..1 of network A shown in Figure 6.1 was subject to five different load conditions. Table 6.1 below shows the interface state at startup, i.e. with no traffic traversing it.

Service Level	Bandwidth			
	Base Allocation	Available	Available with borrowing	Borrowing Capacity
SAAM Control	100	100	100	0
IntServ	300	600	680	120
DiffServ	200	500	620	80
Best Effort	100	100	100	0

Unallocated	300
Total BW	1000

Table 6.1 Interface configuration at startup.

Table 6.2 shows the flow characterization for load A. Different network loads were achieved by changing the flow duration for IntServ, which implicitly affects the projected number of active IntServ flows. Flow requests for each of the DiffServ and IntServ services were generated using a Poisson distribution, and the flow durations modeled with a normal distribution. Table 6.3 summarizes the projected flow requests for the five different loads, with ensuing traffic going from router A to router E.

Parameter	IntServ	DiffServ
Inter-arrival time (sec)	1	10
Duration – mean (sec)	100	200
Duration – sigma (sec)	10	10
Bandwidth (kbps)	6	7

Table 6.2 Characterization of individual flows

For each of the five loads, four different random generation seeds were used. For each seed, the simulation was run twice, first with inter-service borrowing enabled and then with the borrowing disabled. Therefore, there were a total of 8 simulation runs per load. Appendix D contains summary information for each of the described test conditions.

Load	Projected Data			
	Active Flows		Bandwidth Request (Kbps)	
	IntServ	DiffServ	IntServ	DiffServ
A	100	20	600	140
B	104	20	624	140
C	108	20	648	140
D	112	20	672	140
E	116	20	696	140

Table 6.3 Characterization of individual flows

A single simulation run was done over a period of 1000 seconds, which allowed over a 1,2000 flow requests to be processed for both services. Figure 6.4 shows an example of a time-graph plotted from raw data obtained for load A, without inter-service borrowing and with a seed of 100. When all runs were done, the steady-state interval was considered between 250 and 1000 seconds. This was so since in all test cases after 250 seconds service level loads were observed to have reached a steady state.

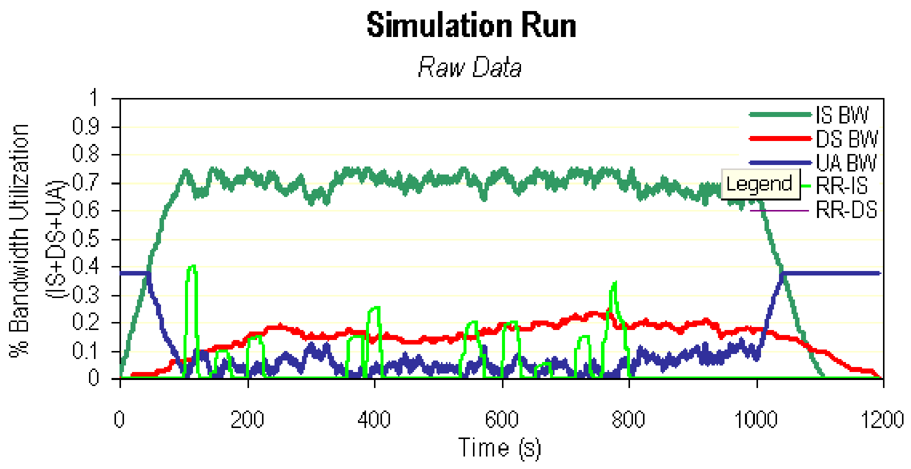
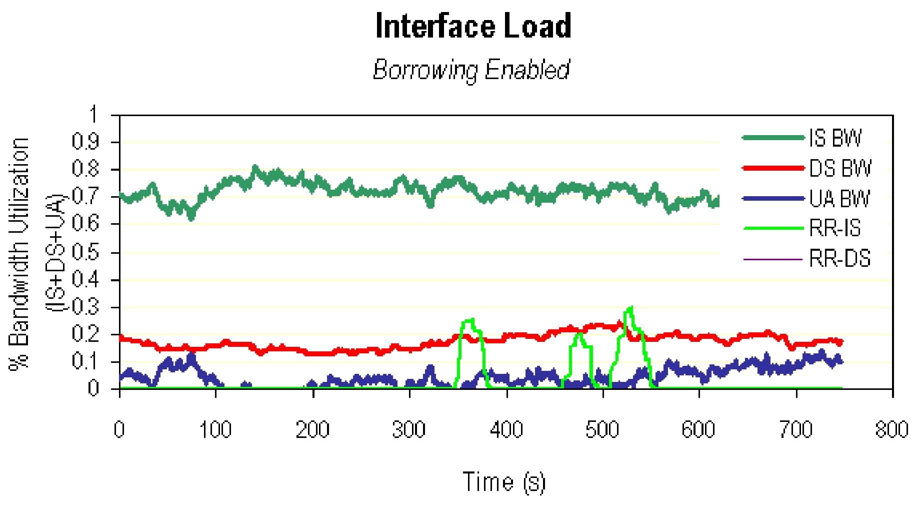


Figure 6.4 Raw data from a single simulation run.

Having defined the steady-state interval for all loads, then the sample respective sample data was extracted from the original data. Figure 6.5 below shows two graphs plotted from data of the same load condition of figure 6.4. The two borrowing states are represented. As can be observed, only the steady-state interval is shown. Each graphs shows the effective interface load per service level (IntServ and DiffServ) plus unallocated bandwidth. Additionally, the rejection rates per service level are also represented.



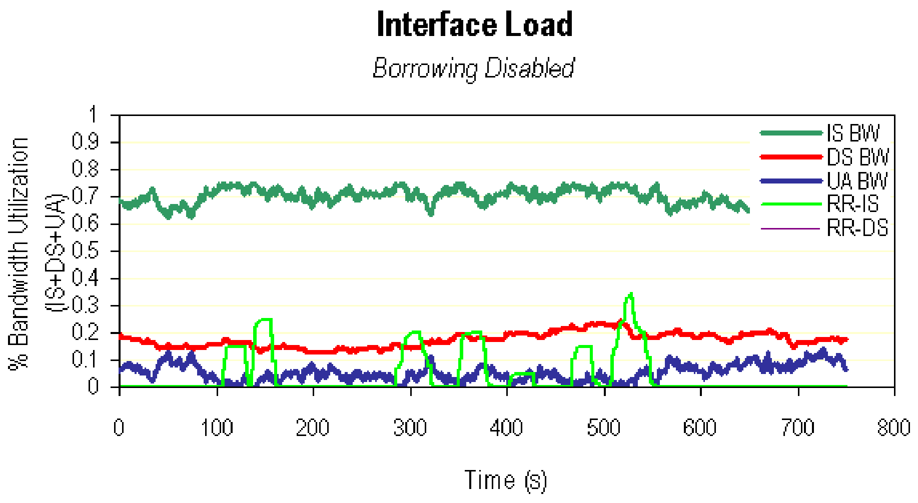


Figure 6.5 Interface 2..1 load during two simulation runs, with and without inter-service borrowing

As previously stated, simulation data details can be found in Appendix D. Graphs in Figures 6.6 and 6.7 summarize the findings. For an increasing load of IntServ and as expected, the flow rejection rate of IntServ also increases. Note that the flow rejection rates were obtained by averaging the results over four runs with different random generator seeds. In all load cases, the impact of inter-service borrowing is extremely significant. It is also noted that the gains of using inter-service borrowing slightly decrease with the increase of the network load which is explained with the saturation of the borrowing capacity made available by the other service.

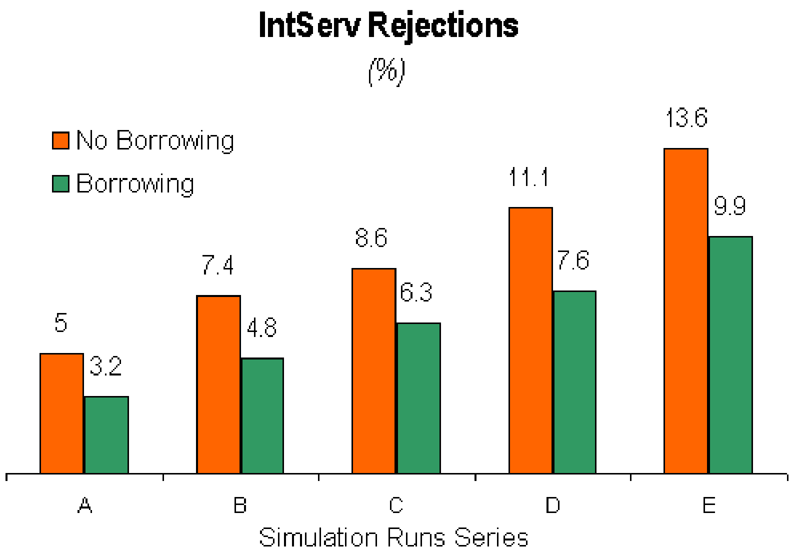


Figure 6.6 Comparative analysis of borrowing versus no borrowing, for increasing network load

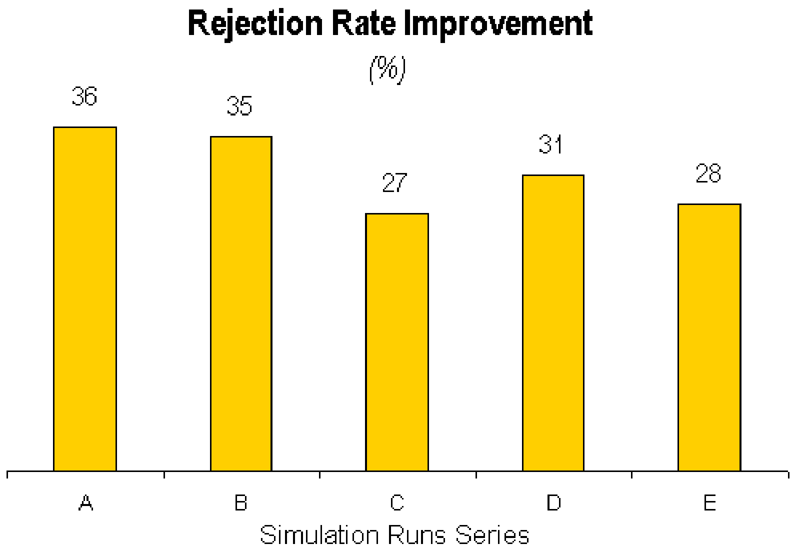


Figure 6.7 Reduction of IntServ flow rejection rate as network load increases

2. Network B Test

For the purpose of this test, the PibTester was adapted to generate flow requests from a random source to a random destination, among all three nodes (A, B and C). Since the test objective was to check the PIB function when more alternate paths are available, there was no need to collect data at the interface level. With the simulation running, the

PibTester GUI, as presented in figure 6.1, offers the option of inspecting the status of individual paths and interfaces at any point during the simulation. This option was used to verify the functionalities of the PIB. The observed results were as expected.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND RECOMMENDATIONS

This thesis demonstrated the feasibility of efficient management of network resources while providing support for different classes of QoS traffic. The novel inter-service borrowing mechanism was developed and integrated with SAAM. This new mechanism results in a more dynamic and adaptive link share among supported services. The test and evaluation results show evidence of a significant improvement of the overall network resource utilization. The new resource management concept further strengthens the goal of SAAM to intelligently manage network resources. Some key aspects of this study are covered below.

A. PATH INFORMATION BASE REDESIGN

The implementation of inter-service borrowing in PIB involved changes in every aspect of the PIB internals. Although some new data members were required to be added to existing data structure, some internal algorithms were completely redesigned with efficiency in mind. One of such modifications is related with processing LSA messages and propagating interface updates. Despite the newly added functionality and associated complexity, the revised PIB is considered more robust and more efficient.

B. PIB TEST

The test drive specifically developed for this thesis proved very valuable. The PIB tester was capable of generating not only a continuous stream of flow requests but also the required Link Status Advertisement messages with the adequate periodicity. The friendly and flexible interface of the tester allowed for a large number of testing conditions and network loads to be run against PIB. For the first time, it was possible to stress test all the functional parts of PIB at the same time. Each test run generated large amounts of data. Processing such large quantities of data, validating them and extracting relevant information was at some point overwhelming for the testing platform. However, obtained results were very satisfactory which helps to strengthen the SAAM concept.

C. AREAS FOR FURTHER STUDY

1. Link State Advertisement Cycle

The LSA cycle duration should not be arbitrarily selected. One of the major processing overheads within PIB is the processing of LSAs. If routers advertise the state of their interfaces very frequently and if the SAAM region contains a large number of routers/interfaces, the burden of processing LSAs may impact the performance of PIB. In such cases, flow requests may have to be buffered while waiting for an LSA cycle to complete. However, a short LSA cycle also means that the PIB state is oftener refreshed, thus maintaining a more accurate image of its network. If otherwise the LSA cycle is made too long, PIB state is less accurate which may eventually cause incorrect admission of new flows, leading to buffer overruns at routers. The LSA cycle need not to be constant and could vary depending upon network conditions. Additionally, some kind of prioritization among flow requests and LSA messages arriving to PIB should be implemented. The tuning of the LSA cycle is therefore an important PIB function and should therefore be in the scope of further study of SAAM.

2. Accurate Link State Advertisement

Whatever modifications are made within the Path Information Base, its function is highly dependent on the external two inputs it receives – the Link State Advertisement and the Flow Request messages. Flow requests simply translate the demand of resources from user applications and is already working as expected. However, LSA generation is not yet fully implemented by SAAM routers. The Link Sate Monitor is the module responsible for monitoring interfaces and advertising their state based on observation of real traffic. Current implementation of the Link State Monitor reports zero interface utilization or other inaccurate value regardless of traffic flows. Once these values arrive to PIB in LSA messages, they are subsequently used to update PIB status. Because of the inaccurate state of PIB, the admission control and resource reservation mechanisms of PIB in most cases will perform incorrectly. In order to make the best use of current functionality of PIB it is highly desirable that LSA generation in SAAM be fully implemented.

3. SAAM Network Management

Every component deployed in a network is typically operated either locally through console access or remotely using network management applications. These applications communicate with network components to query their state or to set functional parameters using an application layer protocol such as Simple Network Management Protocol (SNMP). The PIB within a SAAM server has evolved to a stage where some of its behavior can be changed during normal operation. For instance, the inter-service borrowing capacity can be turned on or off during normal operation. An area of potential study would be the design and implementation of an application that was able to generate SNMP traffic destined to specific SAAM modules/agents such as the SAAM server and the PIB, to be able to perform network management functions.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A MILITARY RELEVANCE OF SAAM PROJECT

A. SUMMARY

The vision for future joint war fighting of US military is described in Joint Vision 2020 (JV2020). The concept of network-centric warfare (NCW), first conveyed in the JV2010 and carried forward in JV2020, represents a fundamental shift from the previous platform-centric warfare. Interoperability with external agencies and among forces of the allied nations is a growing necessity as recently proved with the combined NATO operations in the Balkans. Military operations in the current information age are organized around the NCW concept, through which information superiority translates into increased combat power. NCW is enabled by effectively networking sensors, decision makers and shooters to achieve shared awareness, increased speed of command and high levels of self-synchronization.

The NCW environment creates a wide range of network service requirements, only possible to meet through active and adaptive networks. Server and Agent Based Active network Management (SAAM) is one of such networks being prototyped at the Naval Postgraduate School. This document addresses some key enabler technologies of SAAM, which illustrate the importance of SAAM in the context of the NCW environment.

B. DISCUSSION

Joint Vision 2020 builds on the foundation of Joint Vision 2010. Several strategic principles and operational concepts of JV2020 are technically addressed by SAAM.

1. Global Information Grid

JV2020 develops a concept labeled Global Information Grid (GID). The GID requires a network-centric environment that integrates traditional forms of information operations with sophisticated all-source intelligence, surveillance, and reconnaissance in a fully synchronized information operation. SAAM supports dynamic, non-intrusive service deployment with which software agents can be dynamically deployed across the GID and configured to perform various tasks on demand.

The GID will be a globally interconnected, end-to-end set of information capabilities, associated processes, and people to manage and provide information on demand to warfighters, policy makers, and support personnel. The SAAM hierarchical architecture and auto-configuration protocol provide a mechanism for SAAM to scale from single SAAM regions into a global information infrastructure. Additionally, the quality-of-service (QoS) model of SAAM supports guaranteed services, capable of delay guarantees to individual network users, which is an important guarantee for applications that rely on synchronization.

The GID needs to continue functioning when under hostile attacks. The centralized SAAM approach makes SAAM servers a privileged network player possessing the broadest possible view of distributed GID resources. With such visibility over its resources, SAAM servers are able to take a pro-active approach to fault tolerance by creating optimal alternative paths ahead of time. Under malicious aggression, SAAM immediately redirects affected flows to these alternative paths, which happens seamlessly to network users. The survivability of the GID is further enhanced with SAAM's ability of relocating server functionalities to a different physical network node rapidly without significant service degradation. There is no single point of failure.

2. Innovation

JV2020 identifies technical innovation as a vital component of the revolution in future warfare. SAAM is an agent-based network, which means that new functionality can be easily deployed on the fly. New SAAM agents can extend or replace the functions of existing agents. Once the SAAM network infrastructure is deployed, the introduction of network changes, new requirements, or added functionalities can be easily accommodated.

3. Interoperability

Interoperability is the ability to provide services to and from other systems. It is a mandate for the joint force of 2020, especially in terms of communications, and information sharing. Multinational operations like recent NATO operations in the Balkans, once again demonstrated the importance of interoperability. Information systems and equipment that enable a common relevant operational picture must work from shared networks that can be accessed by any authorized participant regardless of the location.

The centralized approach of SAAM to network management provides for a controlled access to network resources. Identification and authentication of network users is supported and it provides the means for implementing secure communication channels. Additionally, SAAM agents can be tailored to provide the bridge between the varying levels of technology of the potential multinational allied nations. Specially configured agents may be deployed to edge nodes that interconnect incompatible system with the shared network. These agents perform all required traffic adaptation, thus supporting interoperability.

4. Communication Command and Control

NCW requires the coexistence of multiple levels of traffic priority for the Communication, Command and Control (C3) channels. The SAAM QoS model enables applications to specify their QoS requirements, including throughput and delay. SAAM resources are allocated hierarchically and the best performing QoS routes can be easily assigned to the highest priority C3 channels, for instance, between front line units and C3 centers.

SAAM implements a better-fit QoS model for battle. With the per-flow management capability, SAAM can establish different priorities for different flows. Different conversations may have different priorities. Battle scenario is constantly changing, and so is supporting network infrastructure and network users. Network resources are limited. Because battle scenarios are frequently in remote and adverse locations, deployment of a network for supporting NCW in those scenarios requires adequate optimization of such limited resources. The intelligent network management approach of SAAM is perfectly suited for those conditions, since allocation of resources is based on both traffic profile and changing network conditions. Whenever QoS performance of some QoS traffic is affected, SAAM network automatically and dynamically adapts to optimize available resources and, equally important, ensures that high priority traffic is favored whenever network conditions degrade.

C. RECOMMENDATION

The work of this thesis greatly contributes for the improvement of the SAAM concept and further extends the potential of SAAM becoming a solution to all major technical problems posed by NCW.

APPENDIX B PIB SOURCE CODE

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C PIB TESTER SOURCE CODE

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D TEST AND EVALUATION DATA

Te following table summarizes the data obtained from 40 simulation runs.

Simulation Run A

		Inter-Service Borrowing: Disabled	Inter-Service Borrowing: Enabled	Difference	Change
IntServ	Flow Requests	723	723	0	-36 %
	Flow Rejections	37	23	-13.5	
	Flow Rejections (%)	5.0	3.2	-1.8	
	Active Flows (avg)	91.3	93.4	2.1	
	Aggregated Throughput	547 847	560 531	12 684	
DiffServ	Flow Requests	79	79	0	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	20.8	20.8	0	
	Aggregated Throughput	145 758	145 833	75	

Simulation Run B

		Inter-Service Borrowing: Disabled	Inter-Service Borrowing: Enabled	Difference	Change
IntServ	Flow Requests	712	723	11	-35 %
	Flow Rejections	54	35	-18.5	
	Flow Rejections (%)	7.4	4.8	-2.6	
	Active Flows (avg)	92.9	96.1	3.2	
	Aggregated Throughput	557 417	576 664	19 247	
DiffServ	Flow Requests	76	79	2.5	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	21.1	21.1	0.1	
	Aggregated Throughput	147 374	147 800	426	

		Inter-Service Borrowing		Difference	Change
		Disabled	Enabled		

Simulation Run C

IntServ	Flow Requests	704	712	8	
	Flow Rejections	62	46	-16	
	Flow Rejections (%)	8.6	6.3	-2.3	-27%
	Active Flows (avg)	93.9	96.8	2.9	
	Aggregated Throughput	563 421	580 787	17 366	
DiffServ	Flow Requests	78	76	-1.3	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	20.7	21.1	0.4	
	Aggregated Throughput	145 021	147 743	2 722	

Simulation Run D

IntServ	Flow Requests	719	719	0	
	Flow Rejections	80	55	-25	
	Flow Rejections (%)	11.1	7.6	-3.5	-31%
	Active Flows (avg)	94.7	99.4	4.7	
	Aggregated Throughput	568 411	596 332	27 921	
DiffServ	Flow Requests	76	76	0	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	19.9	19.9	0	
	Aggregated Throughput	139 572	139 588	16	

		Inter-Service Borrowing		Difference	Change
		Disabled	Enabled		

Simulation Run E

IntServ	Flow Requests	719	719	0	
	Flow Rejections	98	71	-27.0	
	Flow Rejections (%)	13.6	9.9	-3.8	-28%
	Active Flows (avg)	95.6	100.1	4.5	
	Aggregated Throughput	573 737	600 503	26 766	
DiffServ	Flow Requests	76	76	0	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	19.9	19.9	0	
	Aggregated Throughput	139 572	139 583	11	

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Shelton, General Henry H. – "Joint Vision 2020" – Approved by the Chairman of the Joint Chiefs of Staff JV2020, http://www.dtic.mil/jv2020/, June 2000.

[2] Gibson, John H. and Dao-Cheng, Kuo, "Design of a dynamic management capability for the Server and Agent Based Active Network Management (SAAM) system to support requests for guaranteed Quality of Service traffic routing and recovery", Computer Science Department, Naval Postgraduate School, Monterey, September 2000.

[3] Quek, Henry C., "QoS management with adaptive routing for next generation Internet", Computer Science Department, Naval Postgraduate School, Monterey, March 2000.

[4] Braden, R., Clark, D., and Shenker, S., "Integrated Services in the Internet architecture: an overview", RFC 1633, June 1994.

[5] Floyd, Sally, "Link-sharing and resource management models for packet networks, September 13, 1993.

[6] Devore, Jay L., "Probability and Statistics for Engineering and the Sciences", Fourth edition, Duxbury 1998

[7] Vrable, Dean J. and Yarger, John W., "The SAAM architecture: enabling integrated services", Computer Science Department, Naval Postgraduate School, Monterey, September 1999.

[8] Akkoc, Hasan, "A pro-active routing protocol for configuration of signaling channels in Server and Agent based Active network Management", Computer Science Department, Naval Postgraduate School, Monterey, June 2000.

[9] Kati, Efraim, "Fault-tolerant approach for deploying Server and Agent based Active network Management (SAAM) server in Windows NT environment to provide uninterrupted services in routers in case of server failures(s)", Computer Science Department, Naval Postgraduate School, Monterey, March 2000.

[10] Szcepankiewicz, Peter & Velazquez, Luis, "Authentication in SAAM routers", Computer Science Department, Naval Postgraduate School, Monterey, June 2000.

[11] Jain, Raj, "The Art of Computer Systems Performance Analysis", John Wiley & Sons, Inc., 1991.

[12] Cao, Zhiruo, Wang, Zheng, Zegura, Ellen, "Rainbow Queueing: Fair Bandwidth Sharing Without Per-Flow State".

[13] Kurose, James F. and Ross, Keith W, "Computer Networking", Addison Wesley, 1999

[14] Stoica, Ion and Zhang, Hui, "Providing Guaranteed Services Without Per Flow Management", May 1999.

[15] Stoica, Ion, Shenker, Scott., Zhang, Hui, “Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks”, Sept 1998.

[16] Xie, Geoffrey G. and Lam, Simon L., “An Efficient Adaptive Search Algorithm for Scheduling Real-Time Traffic”, in Proceeding of International Conference of Network Protocols, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Chairman, Code CS
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California
   cseagle@cs.nps.navy.mil

4. Direcção do Serviço de Formação
   Marinha - Portugal
   dsf@mail.marinha.pt

5. Portuguese Naval Attaché
   Washington D.C.
   ponavnlr@mindspring.com

6. Dr. Mari W. Maeda
   Program Manager – DARPA/ITO
   Arlington, Virginia
   mmaeda@darpa.mil

7. Dr. Geoffrey Xie
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California
   xie@cs.nps.navy.mil

8. Dr. Bert Lundy
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California
   blundy@cs.nps.navy.mil

9. Mr. Cary Colwell
   Naval Postgraduate School
   Monterey, California
   colwell@cs.nps.navy.mil

10. LCDR Mónica de Oliveira
    Escola Naval
    pmonica@mail.telepac.pt

11. LCDR Angel Sanjose
    Spanish Navy
    Monterey, California
    aesanjose@hotmail.com

12. LCDR Leonardo da Silva Mattos
    Brazilian Navy
    leonardo_mattos@hotmail.com

13. LT António S. Monteiro
    Naval Postgraduate School
    Monterey, California
    asmontei@nps.navy.mil

14. LT António S. Martinho
    Naval Postgraduate School
    Monterey, California
    martinho@pacbell.net

15. LT Eduardo Bolas
    Naval Postgraduate School
    Monterey, California
    ejbolas@nps.navy.mil

16. LCDR Paulo R. Silva
    Portuguese Navy
    pjrsilva@yahoo.com